5

On Transition Time Testing based on Extended Finite State Machines *

Sijian Zhang[a] and Samuel T. Chanson[b]

[a] Department of Computer Science, University of British Columbia
Vancouver, B.C., Canada V6T 1Z4 *email:* <szhang@cs.ubc.ca>

[b]Currently with the Hong Kong University of Science and Technology
email: <chanson@cs.ust.hk>

A number of methods have been developed to evaluate communication protocol performance based on formal specifications. Most of them assume the specifications are given in the form of Extended Finite State Machines (EFSM). The service times of the transitions are key parameters in the performance analysis and are often assumed to be known apriori. This paper presents a methodology to estimate transition service times by testing the protocol implementation as a black box. An approach to deriving test sequences for transition testing is also proposed.

Keyword Codes: C.2.2; D.2.2
Keywords: Finite State Machine; Formal Specifications; Transition Time Testing

1. Introduction

Formal methods have been increasingly used in the specification, design, implementation and testing of software. The application areas range from communication protocols, distributed systems, and real-time systems to critical software systems in which safety, security and reliability are of prime importance. A number of formal description techniques have been proposed to specify protocols and distributed systems, and three of them, viz, LOTOS, Estelle and SDL have been standardized by ISO and/or CCITT [9, 8, 4]. Estelle and SDL are based on Extended Finite State Machines (EFSM).

A number of papers have been published on performance analysis of communication protocols directly from their formal specifications using analytic techniques or simulation (see for example, [10, 3, 12, 7, 1, 6, 11, 15]). Most of the work is based on EFSMs with the assumption that the transition service times[2] are somehow known in advance.

This paper proposes an approach to obtaining the transition service times of a communication EFSM through *performance testing.* Since performance testing has different meanings depending on the focus of the study, we shall classify performance testing for communication protocols into three categories : transition time testing, saturation testing, and benchmark testing.

*This work was supported in part by the Canadian Institute for Telecommunications Research under the NCE program of the Government of Canada.

[2]The transition service time is the CPU time required to execute a specific transition.

T. Mizuno et al. (eds.), *Protocol Test Systems*

Each of the above tests different performance aspects of the protocol implementation under a different work load environment. The objective of *transition time testing* is to obtain the transition service times with respect to the EFSM on which the protocol is based; *saturation testing* aims at measuring the peak performance of a communication protocol under the condition that a specific service center is saturated; and *benchmark testing* is used to predict the system's performance under a specific work load and hardware/software platform. In this paper, only *transition time testing* is discussed. As mentioned above, transition service times are needed as input parameters by most analytic models and simulation packages.

A major problem that makes performance testing difficult is that the Implementation Under Test (IUT) is often given as a black box so that inserting measurement points is almost impossible. Moreover, some transitions in an EFSM can be invisible in that only their inputs or outputs (but not both) are observable. To make matters worse, often a transition can produce a variable number of output messages and the transition can still be in progress at the time its first output message is seen from the outside; or the transition may have finished before its first output message is observed. It should be obvious from the above discussion that measuring the transition times accurately is not a straightforward matter. This paper presents a methodology in measuring/computing the transition service times effectively.

The rest of the paper is organized as follows. Section 2 discusses the relationship between service times and response times and proposes a technique which we shall call the t-test (standing for transition termination test) to estimate the service times based on response times. Section 3 studies techniques dealing with invisible transitions. Section 4 presents a framework to generate test sequences for transition time testing which is followed by an example. Finally, Section 5 concludes the paper.

2. Testing considerations (response time and service time)

The basic idea of performance testing is to send messages to the IUT and measure its response times which are analyzed to evaluate the system's performance. The IUT is assumed to be correct with respect to the specification (e.g., one that has passed conformance testing). Although the protocol implementation is given as a black box, we assume that the upper layer and the lower layer interfaces of the protocol are accessible. A common architecture for performance testing is shown in Figure 1, where a *tester* simulates the peer entity which communicates with the IUT.

A common formalism to specify communication protocols is the *finite state machine* (FSM) or the *extended finite state machine* (EFSM). Figure 2 (from [5]) presents an example of the EFSM for the X.25 LAPB protocol. *The transition and the state* are the two basic constructs in EFSMs. A transition usually consists of input and output events and data processing. The communication protocols can be viewed as executing a sequence of transitions (represented by the arrows in the EFSM) while moving from state to state (represented by the circular nodes).

Conceptually, the state changes after the system has received an incoming message from the environment and finished execution of the transition associated with the message. Therefore, the message sending intervals to the IUT are crucial in performance testing.

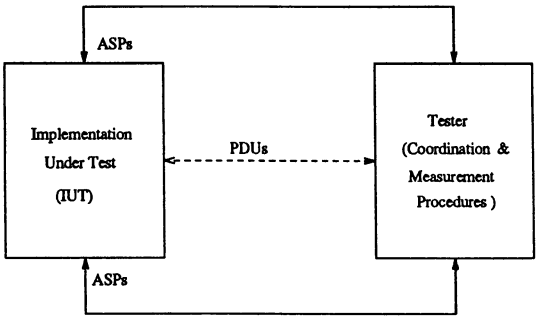

Figure 1: A sample conceptual testing architecture

This is further explained in the following with the help of Figure 3.
Let

- t_i - service time of message i ;
- $\hat{t}_i$ - response time of message i;
- D_i - transmission delay of message i from the tester to the IUT;
- W_i - queue wait time of message i;
- δ_i - latency of message i.

The *service time of a message* is defined as the service time of the transition invoked by the message which includes only the CPU time but no queue wait time. The *response time of a message* is defined as the interval between the time when the message is sent and the time when the first response (usually an outgoing message) is seen. For simplicity, we also use the *response time of a transition* to refer to the response time of the message that invoked the transition. Note that some transitions may not produce any outgoing message and thus the response times of these transitions cannot be measured directly. This will be discussed in the next section.

The *transmission delay of message i* is the time interval between the moment message i is sent by the tester and the moment it arrives at the IUT. The *queue wait time of message i* is the time which the message spends on waiting for service[3]. The *latency of a message* is defined as the interval from the time when the message is sent by the tester to the time when the corresponding transition begins to be executed. Therefore, as shown in Figure 3, we have

$$\delta_i = D_i + W_i \,. \tag{1}$$

The response time $\hat{t}_i$ can be measured directly and accurately, but not so for the transition service time t_i because the transmission delay and the internal queue wait time for service are difficult to obtain precisely if the IUT is given as a *black box*. Therefore, we shall try to estimate the transition service time from the corresponding response time.

[3]To service an incoming message means to execute the transition associated with the message.

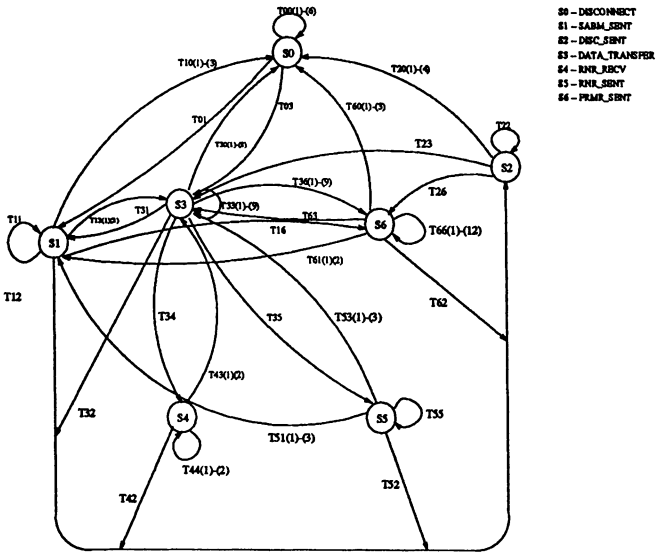

Figure 2: A Finite State Machine (FSM) for X.25 LAPB

In general, $\hat{t}_{i+1}$ will decrease if δ_{i+1} decreases. When the message sending interval τ_{i+1} increases, the waiting time W_{i+1} of transition $(i+1)$ will decrease, which in turn will reduce δ_{i+1} and therefore $\hat{t}_{i+1}$.

In order to obtain the service time of transition i accurately, the interval τ_i of sending messages from the tester should be increased until $\hat{t}_{i+1}$ will not reduce any further, this means the waiting time W_{i+1} of transition $(i+1)$ is zero (i.e., processing of the previous messages is completed by the time message i+1 arrives). In this case, $\delta_{i+1} \approx D_{i+1}$. This situation is illustrated in Figure 4. D_{i+1} is determinable by the tester because the test environment is controllable. D_{i+1} is assumed to be negligible if the tester and the IUT are

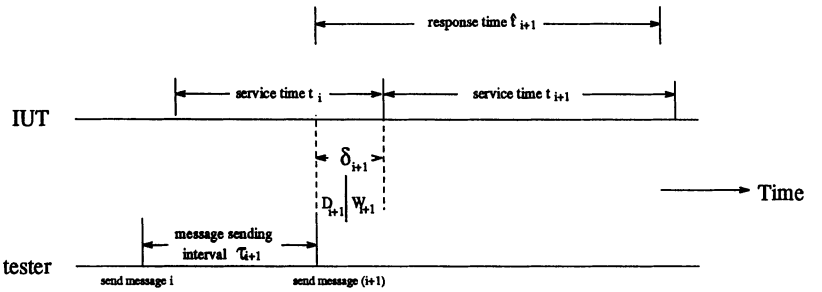

Figure 3: A sample time diagram of message sendings and executions

on the same computer and there is no other job with higher priority running concurrently with the IUT[4]. This means that the technique is most useful when the tester and the IUT are on the same computer or in the same location connected by a short and dedicated link where network load variation is not a problem. Under this condition, and assuming the response time and the service time end at the same time (we shall relax this assumption later on),

$$\hat{t}_i \approx t_i \,. \tag{2}$$

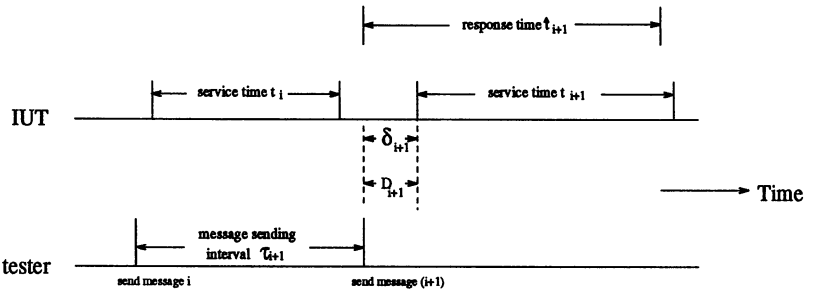

Figure 4: A sample time diagram where there is no transition queue wait time

From this analysis, we see that the tester should control not only what to send to the IUT but also when to send them. When there is no queue wait time, the response time of a message provides a rough estimate of the service time of the corresponding transition.

The above discussion also shows that the response time can be used to determine if the processing of the previous transition has completed. This is useful if the response time of the previous transition cannot be directly observed externally, or if we would like to have a more accurate estimate of service times when the response time and the service time do not end at the same time. The technique is explained below.

Suppose message i is sent at time A as shown in Figure 5; and at time B, the first response due to message i is seen. Let the processing of transition i finishes at time C (C may be before or after B, and C may not be observable externally to the IUT because of the black box assumption). Our objective is to determine C which will allow us to derive the service time of transition i (i.e., AC).

We select time E' after the last response to message i is seen to send the next message (i.e. $i+1$) to the IUT. To determine if at E' the IUT has finished processing the previous transitions, we repeat the sequence of message transmissions except that in the next round, message $i+1$ is sent at time E'' where the interval between B and E'' is twice as long as the one between B and E'. Increasing the sending interval will reduce or eliminate

[4]In this case, the value is usually very small compared to the other times.

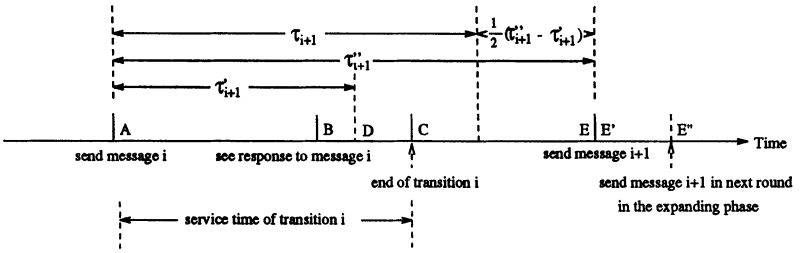

Figure 5: Selecting the message sending intervals

the queue wait time of message i+1 and therefore the response time of message $i+1$ can only decrease (or remains unchanged). If there is no change in the response time, it means the IUT has finished processing of message i and its associated transition by time E'. If not, the previous step is repeated using the same test data but moving E' to E'' and selecting a new value of E'' as before (i.e., $BE' = 2BE$), until there is no change in response time. Without loss of generality, let us call the time instant determined in the round before the last one E, and the smallest response time obtained R_{i+1}. The value of E in the second last round is used because the response times in that round and the last round are the same. Taking the value of the second last round will reduce the overhead in the next phase of testing (see below). The interval between A and E is recorded as τ_{i+1}. Based on the previous discussion, it is obvious that R_{i+1} is the smallest response time for message i+1. This phase is called the *expanding phase* because E' is moved further and further from A. This is achievable because the testing environment is completely under our control so that on other jobs will interfere with the testing. The queue wait time will be zero when the message sending interval is larger than the service time of the previous transition.

Now we are in a position to determine the instant C. We know that $E > C$. The idea is to repeat sending messages i and i+1, but in each round bring E closer to C until the time difference between two successive sendings of message i+1 is less than a predetermined margin. The interval between C and E is less than this difference which is the error margin we are willing to tolerate. Note that all times in a round are measured relative to A.

The procedure is formalized in the following algorithm. The function τ is defined as $\tau(x_2, x_1) = |x_2 - x_1|$. The parameters τ_{i+1} and R_{i+1} are initialized with the values obtained in the expanding phase.

Algorithm : *determining transition completion time after the expanding phase has been completed*

Inputs : 1) pr_i — the preamble[5] of transition i;
2) ps_{i+1}— the postamble[6] of transition i+1;
3) τ_{i+1} — the interval between sending message i and message i+1 ;
4) R_{i+1} — the smallest response time of message i+1;

5) ϵ — the predetermined error margin ;

Outputs : t_i — the service time of transition i ;

Procedure :

1. set $\tau''_{i+1} = \tau_{i+1}$ and $\tau'_{i+1} = \frac{1}{2}\tau_{i+1}$;
2. send pr_i to set the protocol at the starting state of transition i;
3. send message i and label the sending instant as A;
4. send message i+1 at time D where $\tau(D, A) = \tau'_{i+1}$;
5. measure the response time of message i+1 and call it R'_{i+1};
6. if $(R'_{i+1} = R_{i+1})$ then
   set $\tau'_{i+1} = \frac{1}{2}\tau'_{i+1}$;
   go to Step 2;
   else[7]
   continue onto next step (Step 7);
7. set $\tau_{i+1} = \tau'_{i+1} + \frac{1}{2}\tau(\tau''_{i+1}, \tau'_{i+1})$;
8. send pr_i to set the protocol at the starting state of transition i;
9. send message i and label the sending instant as A;
10. send message i+1 at time D where $\tau(D, A) = \tau_{i+1}$;
11. measure the response time of message i+1 and call it R'_{i+1};
12. if $(R'_{i+1} = R_{i+1})$ then
    set $\tau''_{i+1} = \tau_{i+1}$;
    else
    set $\tau'_{i+1} = \tau_{i+1}$;
13. if $(\tau(\tau''_{i+1}, \tau'_{i+1}) <= \epsilon)$ then
    set $t_i = \tau_{i+1}$;
    stop;
    else
    send ps_{i+1} to set the protocol back at the *idle* state;
    go to Step 7.

The first 6 steps of the algorithm try to find a lower bound of the service time. An upper bound has been obtained at the end of the expanding phase. In each round from Steps 7 to 13, the difference of the lower bound and the upper bound is reduced by half to form the new upper bound; this is actually a binary search. So the distance between C and D can be made arbitrarily small. In this way, the service time of message i (i.e., time interval AC in Figure 5) can be determined accurately. This procedure can be automated and shall be referred to as the *shrinking phase* because the upper bound is reduced in this

[5]The preamble of a transition i is a sequence of transitions starting from the *idle state* and leading to the start state of transition i.

[6]The postamble of a transition j is a sequence of transitions bringing the protocol from the end state of transition j back to the *idle* state.

[7]Note that R'_{i+1} cannot be less than R_{i+1} because R_{i+1} is the smallest response time obtained in the expanding phase.

phase. The expanding phase and shrinking phase together constitute what we call the *t-test.*

As shown above, the complete t-test needs to execute the same test subsequence a number of times. The number decreases if we are willing to accept a larger error in service time estimation. If it is adequate to use the response time of a transition as an estimate of its service time in performance analysis, then only the expanding phase of the t-test is needed to make sure no queue wait time occurs for the specific transition. The expanding phase of the t-test is much easier to achieve and in the best case it only needs two rounds. In the rest of the paper, we assume only the expanding phase of the t-test is used in the transition time test whenever the response time of a transition can be measured directly.

3. Visible and invisible transitions – discussion of testability

The t-test requires at least two consecutive transitions. It assumes that the input of the first transition is controllable and both the input and the output of the second transition are observable. However, not all transitions in an EFSM have observable input or output events. Transitions without any input are known as *spontaneous transitions* [14]. We shall define a transition without any input (i.e. -/O) or a transition without any output (i.e. I/-) as an *invisible transition.* Conversely, *a visible transition* is defined as a transition in which both its input and output are observable. A transition dealing with the timeout event is a typical invisible transition because timeout is usually generated internally.

Invisible transitions make performance measurement difficult. Their response times can not be obtained directly and thus can not be used to detect the completion of previous transitions. Fortunately, sometimes this problem can be solved by combining an invisible transition with other transition(s). This will be discussed in detail later in this section.

Figures 6 and 7 list all the possible situations of two consecutive transitions involving invisible transitions. Each of the cases is analyzed below :

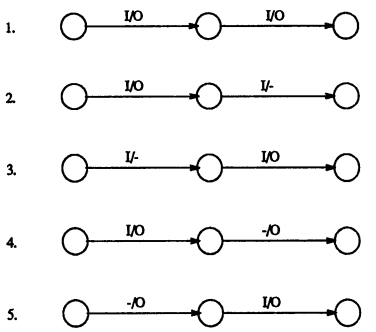

Figure 6: Two consecutive transitions with at least one visible transition

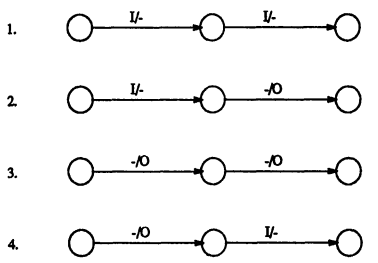

Figure 7: Two consecutive transitions without any visible transition

1. The simplest case is when both transitions are visible (Figure 6 (1)). The measured response time can be used to estimate the service time of the corresponding transition. Furthermore, as explained in Section 2, the response time of the second transition can also be used to determine the service time of the first transition accurately by the t-test.

2. In Figure 6 (2), the second transition is invisible and its response time cannot be measured directly. Therefore, it cannot be used directly to determine the service time(s) of its preceding transition(s). However, if the first transition can be combined with other visible transitions behind it as shown in path 2 of Figure 8 (1), then we have case 1 and the service time of the first transition can be accurately determined. Furthermore, the second transition can also be treated as a visible one by combining it with a visible transition that follows it (path 1 in Figure 8 (1)). Path 1 then becomes case (3) in Figure 6 and can be analyzed using the technique described below.

3. The second transition in Figure 6 (3) is visible. T-test can be used to determine the service time of the first transition (which is invisible) as discussed in Section 2.

   Alternatively, if the two transitions in Figure 6 (3) are treated as one transition, it is visible and its response time can be obtained directly. Suppose the response time is t and the response time of the second transition is t_2. Therefore, the response time of the first transition, t_1 can be obtained using the simple relation : $t_1 = t - t_2$. t_1 can be used as a rough estimate of the service time of the first transition.

4. Figure 6 (4) can be handled in a way similar to that of Figure 6 (3). The second transition is *invisible* because it can be fired without any input. However, we can combine these two transitions and treat them as one visible transition as discussed in case 3.

5. The first transition of Figure 6 (5) is not useful and neither is the combination of the two transitions because it is still invisible. However, it may be possible to calculate

the service time of the first transition with the help of the second visible transition behind it and some simple computation as discussed later in this section.

6. In cases (1), (3) and (4) of Figure 7, like the case in Figure 6 (5), the two transitions are not very useful either individually or as a unit. Other transitions in the EFSM are needed to compute their service time (see below).

7. The two transitions in Figure 7 (2) can be combined into one *visible* transition if necessary. However, the individual service time for each transition can only be obtained if at least one of them can be measured through other subpaths.

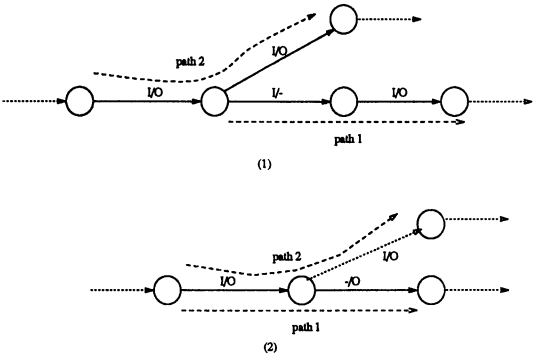

Figure 8: Example of test subpaths for invisible transitions

Sometimes, it is necessary to combine two or more consecutive transitions in the EFSM as one transition for use in the test. We shall call the combined transitions a *compound transition.* A compound transition is visible if the input of its first transition and the output of its last transition are both observable. For example, $I_1/-, \cdots, -/O_n$ is a visible compound transition.

A visible (compound) transition is always *measurable.* Its service time can be estimated by measuring its response time. We say a (compound) transition is *testable* in transition time testing if its service time can be obtained either by direct or indirect measurement. By this definition, a visible transition (either single or compound) is always testable. An invisible (compound) transition may become testable under certain circumstances as discussed earlier.

It is reasonable to assume that in a communication protocol, there exists at least one observable input event and one observable output event. Furthermore, since a correct EFSM is always strongly connected, the following proposition is true :

Proposition 3.1 *An invisible transition can always be included either at the head or at the tail of a visible compound transition.*

An invisible (single) transition has the form I/- or -/O. If the invisible transition is of the form I/-, the visible compound transition is the transition path starting with I/- and ending with the transition having an observable output event. If the invisible transition has the form -/O, the visible compound transition is the transition path that starts with an observable input event, and ending in -/O.

To tackle invisible transitions, we present the following theorems.

Theorem 3.1 *An invisible transition is* testable *if it can be included as a testable compound transition and the rest of the compound transition is testable.*

Proof: It is obvious from the definition of testable transition. □

Theorem 3.2 *(1) An invisible transition I/- is testable if there is at least one testable (compound) transition immediately after it in the EFSM which begins with I/O or I/-; (2) An invisible transition -/O is testable if there is at least one testable (compound) transition immediately before it in the EFSM which ends with I/O or -/O.*

Proof:

(1) Let us consider the case of I/-.

If the testable compound transition succeeding I/- begins with the transition I'/O', we have case 3 of Figure 6 and the service time of I/- can be derived using the t-test. If the testable compound transition begins with I'/-, according to Proposition 3.1, there is always a visible compound transition which begins with I'/-. Furthermore, this visible compound transition prepended with I/- is also visible. From Theorem 3.1, transition I/- is also testable.

(2) Similar analysis can be applied to prove (2) of the theorem. □

The following corollary is a special case of Theorem 3.2.

Corollary 3.3 *(1) An invisible transition I/- is testable if there is a transition immediately after it in the EFSM which is of the form I/O or I/-; (2) An invisible transition -/O is testable if there is a transition immediately before it in the EFSM which is of the form I/O or -/O.*

In the next section, we present a method for selecting test sequences for transition time testing based on the above discussion of testability.

4. Test sequence generation

The objective of *transition time testing* is to obtain the service times of all the transitions of the EFSM on which the protocol is specified. When the service times are determined, the system's performance can be analysed by using *queueing analysis* [15], *reachability analysis* [12, 2], or other methods [10, 11]. Therefore, all the transitions should be covered in the test sequences. Furthermore, the test sequences should ensure that there is no queue wait time for the visible (compound) transitions so that

$$\hat{t}_i \rightarrow t_i, \tag{3}$$

as discussed before. Therefore, a test subsequence should include at least one visible (compound) transition. As explained earlier, the visible (compound) transition can be used to estimate its own service time and/or to determine the service time of its previous transition in the t-test.

The framework for generating test subsequences for transition time testing consists of two phases. In phase one, all the *visible* transitions in the EFSM are selected to be tested and their service times obtained. In phase two, the other transitions left (mostly *invisible*) are considered in the way presented in the previous section.

To do so, the EFSM of the protocol is considered as a directed graph, denoted as G, where the nodes stand for the states and the directed edges for the transitions. G is a strongly connected graph because, for any correct communication protocol (specified by an EFSM), there should be at least one path leading to any state from the *idle* state, and at least one path bringing the protocol back to the *idle* state.

In the derivation of the test sequences, a subgraph of G is first constructed, denoted as $G^{(0)}$. It includes all the nodes and all the visible transitions of G (excluding the invisible transitions). The set $\mathcal{S}^{(0)}$ initially contains all the invisible transitions of the EFSM. $G^{(0)}$ may not be a strongly connected graph any more as shown in Figure 9. However, all the edges in $G^{(0)}$ are visible and testable. So each transition in $G^{(0)}$ is treated as a subsequence by itself.

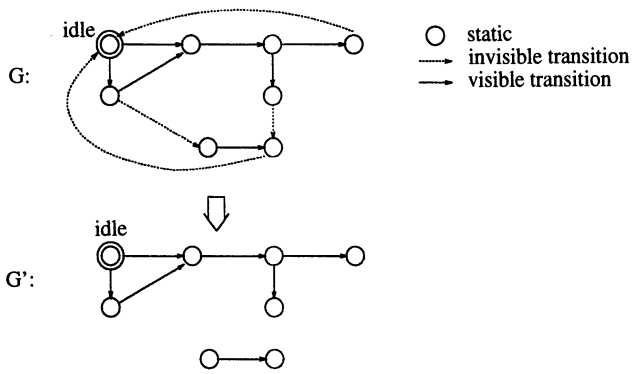

Figure 9: A sample EFSM graph showing disconnected subgraphs after deleting invisible transitions

Phase two begins after the test subsequences for all the visible transitions have been generated. Select an invisible transition from $\mathcal{S}^{(0)}$ and put it into $G^{(0)}$. If the selected invisible transition is of the form I/-, check if there is any transition in the current graph immediately after it. If not, put it back into $\mathcal{S}^{(0)}$ and pick another transition. If the following transition is of the form I_1/O_1, the test subsequence (I/-, I_1/O_1) is generated; otherwise, if the following transition is of the form -/O, select the test subsequence (I/-,-

/O). If the above cases are all false, the following transition must be of the form I_1/-. In this case, two visible compound transitions are generated, one starting with I_1/- and the other with (I/-, I_1/-). Note this is always possible since all transitions in $G^{(0)}$ are testable. The difference in service times of the two compound transitions gives an estimate of the service time of I/-.

Likewise, if the selected invisible transition is of the form -/O, check if there is any transition in the current graph immediately before it. If not, put it back into $\mathcal{S}^{(0)}$ and pick another one. If the transition preceding -/O is of the form I_1/O_1, the test subsequence (I_1/O_1, -/O) is generated; otherwise, if the preceding transition is of the form I/-, the compound transition (I/-, -/O) is selected as a test subsequence. If both are false, the preceding transition must be of the form -/O_1. In this case, two visible compound transitions are generated, one ending with -/O_1 and the other with an (-/O1, -/O).

If the newly selected invisible transition becomes testable, it remains in $G^{(0)}$, and the graph becomes $G^{(1)}$ with one more edge while $\mathcal{S}^{(0)}$ becomes $\mathcal{S}^{(1)}$ with one fewer transition. This process is repeated until the set of invisible transitions is empty or none of the invisible transitions in the set can be made testable using this procedure.

Finally, we reconsider all the transitions remaining in $\mathcal{S}^{(n)}$. First, for each transition of the form I/-, check in G if there is another transition of the form $I_k/-$ following it, which may still be in $\mathcal{S}^{(n)}$. If so, this transition can become testable according to Corollary 3.3. Two visible compound transitions are then generated, one starting with $I_k/-$ and the other with $I/-, I_k/-$. Record these two subsequences for the transition, remove it from $\mathcal{S}^{(n)}$ and put the transition in $G^{(n)}$. Similarly, each invisible transition of the form -/O in $\mathcal{S}^{(n)}$ is checked. The procedure ends when all the transitions in $\mathcal{S}^{(n)}$ have been checked or the set is empty.

If any invisible transition has become testable in this phase, Phase Two of the procedure is repeated until no remaining transitions can be made testable.

Note that it is possible some transitions in the EFSM are untestable. Figure 10 shows an example EFSM with two untestable transitions (I/- and -/O). How to deal with these transitions is left as future work.

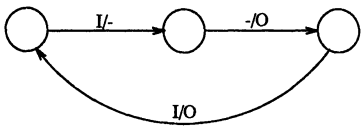

Figure 10: A sample EFSM with untestable transitions

Each test subsequence derived above is a visible compound transition. To form the test sequences, the preamble which is the transition path starting from the *idle state* and leading to the start state of the compound transition, and the postamble which is the transition path bringing the protocol from the end state of the compound transition back to the *idle* state must be added. Since a test subsequence may need to be tested many

times, the shortest preamble and postamble should be used. The algorithm to find the shortest path between two vertices in a directed graph can be found in [13].

An example

To illustrate, we use the abstract EFSM shown in Figure 11.

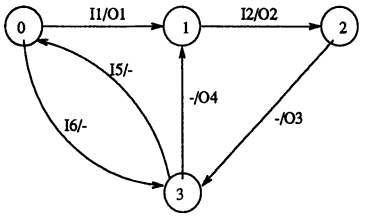

Figure 11: An example EFSM to show test subsequence generation

Suppose that Figure 11 is graph G. The initial subgraph $G^{(0)}$ of G in the test sequence generation procedure is given in Figure 12 (a). The initial invisible transition set is $\mathcal{S}^{(0)} = \{-/O3, -/O4, I5/-, I6/-\}$. Two test subsequences are generated from $G^{(0)}$: I1/O1 and I2/O2.

Next, the first invisible transition in $\mathcal{S}^{(0)}$, -/O3, is selected and put in $G^{(0)}$. It can become testable using the subsequence I2/O2, -/O3. $G^{(1)}$ is given in Figure 12 (b) and $\mathcal{S}^{(1)} = \{-/O4, I5/-, I6/-\}$.

Next, -/O4 is selected because -/O3 is now testable. The test subsequence for -/O4 is I2/O2, -/O3, -/O4. $G^{(1)}$ is given in Figure 12 (c) and $\mathcal{S}^{(2)} = \{I5/-, I6/-\}$.

I5/- can also become testable using the t-test with the test subsequence (I5/-, I1/O1), and $\mathcal{S}^{(3)}$ becomes $\{I6/-\}$. Finally, (I6/-, -/O4) is used to derive the service time of I6/-.

In summary, the test subsequences derived are :

```
(1) I1/O1;
(2) I2/O2;
(3) I2/O2, -/O3;
(4) I2/O2, -/O3, -/O4;
(5) I5/-, I1/O1;
(6) I6/-, -/O4;
```

5. Conclusion

We have shown that it is possible to obtain the service time of a transition using the t-test. The t-tests need to execute the same subpath more than once in order to make sure the sending interval is long enough for the previous transitions to finish processing. Although some transitions are invisible, it may still be possible to determine their service times by combining them with other transitions to form testable compound transitions.

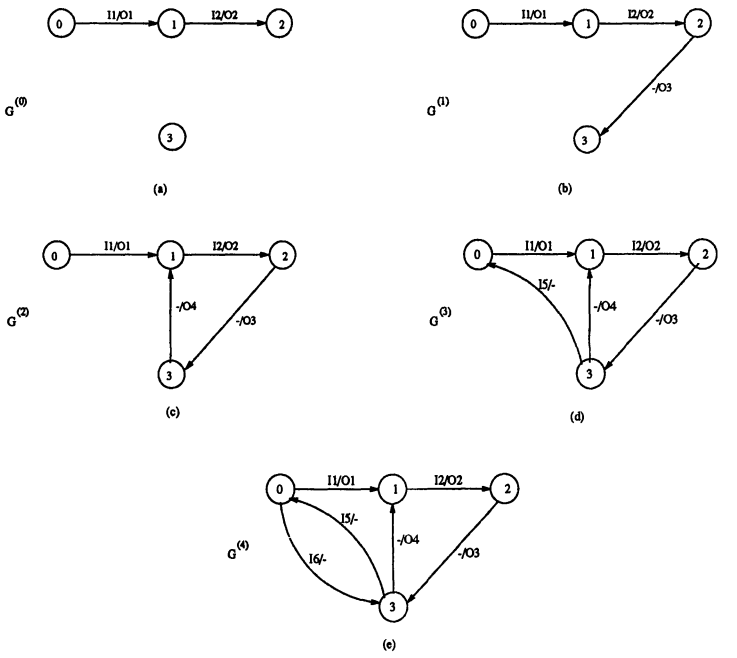

Figure 12: Test Subsequence generation procedure for the example given in Figure 11

The key to *transition time testing* is that the tester must have control on both the test sequences and the sending intervals of the messages. This paper has analyzed and proposed a scheme for test sequence generation and estimation of transition service times given the EFSM of the protocol.

In test subsequence generation, the best case is when all the transitions of an EFSM are visible. In this case, only Phase one of the generation procedure needs to be executed and its time complexity is $O(n)$ where n is the number of the transitions. We know that each round of Phase two will take $O(nm + m^2)$ where m is the number of invisible transitions remaining in the set. Furthermore, Phase two will have to be repeated if any invisible transition becomes visible in the round. Therefore, the worst case situation occurs when only one invisible transition becomes visible in every round so that Phase two has to be repeated the maximum number of times. The time complexity of the worst case is $O(nm^2 + m^3)$ where m is the number of invisible transitions.

REFERENCES

1. Guido Albertengo, Silvio Forno, and Andrea Fumagalli. TOP/PDT: A Toolkit for the Development of Communication Protocols. *IEEE Journal on Selected Areas in*

Communications, 8(9), 12 1990.

2. Falko Bause and Peter Buchholz. Protocol Analysis using a timed version of SDL. In *Formal Description Techniques, III (IFIP)*. Elsevier Science Publishers B. V. (North-Holland), 1991.
3. G.v. Bochmann and J. Vaucher. Adding performance aspects to specification languages. In *IFIP Symposium on Protocol Specification, Testing and Verification VIII*, pages 19–31, Atlantic City, 7 1988. Elsevier Science Publishers B. V. (North-Holland).
4. CCITT. *Specification and Description Language – Recommendation Z.100.* Geneva, Switzerland: CCITT press, 1986.
5. Samuel T. Chanson and Sijian Zhang. A Test Case Management System. In *Particpants' Proc. of 5th International Workshop on Protocol Testing Systems*, Montreal, Canada, 1992.
6. D. Fernandez, E. Vazquez, and J. Vinyes. Io: An Estelle Simulator for Performance Evaluation. In *FORTE'91*, 1991.
7. Jan Gustafsson and Harry Rudin. Including a Queue in a Formal Description Driven Protocol Performance Analysis. In *9th IFIP WG6.1 Intnl. Symp. on Protocol Specification, Testing and Verfication*, Enschede, The Netherlands, 1988. Elsevier Science Publishers B.V. (North-Holland).
8. ISO TC 97/SC 21. *Information processing systems – Open systems Interconnection – Estelle (Formal Description Technique based on an Extended State Transition Model.* ISO 9074. Interational organization for Standardization, 1989.
9. ISO TC 97/SC 21. *Information processing systems – Open systems Interconnection – LOTOS – A formal description technique based on the temporal ordering of observational behaviour.* ISO 8807. Interational organization for Standardization, 1989.
10. Pieter S. Kritzinger. Protocol Performance Using Image Protocols. In H. Rudin and Colin H. West, editors, *IFIP Symposium on Protocol Specification, Testing and Verification VII*, Zurich, 5 1987. Elsevier Science Publishers B. V. (North-Holland).
11. Pieter S. Kritzinger and Graham Wheeler. Semi-Markovian Analysis of Protocol Performance. In P. Wolper A. Danthine, G. Leduc, editor, *Protocol Specification, Testing and Verification, XIII.* IFIP, 1993.
12. Fuchun Joseph Lin and Ming T. Liu. An Integrated Approach To Verification And Performance Analysis of Communication Protocols. In *IFIP Symposium on Protocol Specification, Testing and Verification VIII*, Atlantic City, 7 1988. Elsevier Science Publishers B. V. (North-Holland).
13. Kenneth A. Ross and Charles R.B. Wright. *Discrete Mathematics.* Prentice Hall, 1988.
14. B. Sarikaya. *Test design for computer network protocols.* PhD thesis, McGill University, Canada, 3 1984.
15. Sijian Zhang and Samuel T. Chanson. An Approach to Evaluating the Performance of Communications Protocols based on Formal Specifications. In *Proc. of IEEE International Conference on Netowrk Protocols*, San Fransico, USA, 10 1993.