Detecting faults in TFTP implementations using Finite State Machines with timeouts

M. Zhigulin, S. Prokopenko, M. Forostyanova
Radio-physical department
Tomsk State University
Tomsk, Russia
maxzh81@gmail.com, s.prokopenko@sibmail.com,
mariafors@mail.ru

***Abstract*—In this paper, we consider a test derivation strategy for testing protocol implementations based on Finite State Machines with timeouts. The strategy is applied for testing TFTP implementations.**

Keywords-Finete State Machine (FSM); FSM with timeouts (timed FSM); transition tour

I. INTRODUCTION

FSM-based test derivation strategies for conformance testing of protocol implementations are well known [see, for example, 1-3] and a number of formal methods were developed for deriving tests which check time constraints of a discrete event system implementation. Some of them use FSM-based strategies for test derivation [4-11]. One of such strategies uses the model of a timed FSM (TFSM) with so-called timeouts [7-11], i.e., if no input is applied during an appropriate time period the FSM can move to another prescribed state. Correspondingly, the behavior of an FMS significantly depends on a time instance when an input is applied, i.e., the behavior of the FSM is specified for timed input sequences. In [12], it is shown how this behavior can be described by an ordinary FSM with an additional input symbol (a time unit) and thus, despite of the fact that a test suite derived for such an abstract FSM using black-box testing methods returns highly redundant tests, FSM-based test derivation methods can be directly used when deriving tests from an FSM with timeouts. In this paper, we derive a test suite as a transition tour of an appropriate FSM, since W-based testing methods [3] ask for the specification FSM to be complete and deterministic and this usually does not hold for FSMs which describe protocol behavior. We then analyze the fault coverage of a transition tour for TFTP (Trivial File Transfer Protocol) [13] implementations where the behavior significantly depends on timeouts. The contributions of the paper can be summarized as below.

- In this paper, we manually extract an FSM with timeouts from the description of TCTP, transform the obtained TFSM into a classical FSM and derive a test suite as a transition tour of the latter.
- The obtained test suite is then applied to available TFTP implementations and the set of detected faults is described.

The structure of the paper is as follows. Section 2 contains the preliminaries for FSMs with timeouts (TFSMs) and a brief description of the transformation of such TFSM into a classical FSM. In Section 3, the TFCP behavior as a timed FSM is presented and the set of faults which were detected in protocol implementations using a derived test suite is described. Finally, we conclude the paper in Section 4.

II. FINITE STATE MACHINES WITH TIMEOUTS

In this paper, we consider Finite State Machines (FSMs) which are augmented with timeouts in order to explicitly take into account timed aspects of the system behavior. A *timed FSM* $\mathcal{S}$ (or TFSM) is a 6-tuple $(S, I, O, \lambda_S, s_0, \Delta_S)$, where S, I and O are finite non-empty sets of states, inputs and outputs, respectively, s_0 is the initial state, $\lambda_S \subseteq S \times I \times O \times S$ is a transition relation, Δ_S: $S \to S \times (N \cup \{\infty\})$ is a *timeout function* where N is the set of nonnegative integers.[1] The function Δ_S has two projections $\Delta_S(s)_{\downarrow N}$ and $\Delta_S(s)_{\downarrow S}$. If no input is applied at a current state s before a timeout $\Delta_S(s)_{\downarrow N}$ expires then the TFSM will move to another state $\Delta_S(s)_{\downarrow S}$ as it is prescribed by the timeout function. If $\Delta_S(s)_{\downarrow N} = \infty$ then the TFSM can stay at the state s infinitely long waiting for an input, i.e., in this case, $\Delta_S(s) = (s, \infty)$. As usual, the notation $s - i/o \to s'$ is used for a 4-tuple $(s, i, o, s') \in \lambda_S$ while using $s - t \to s'$ for a triple $\Delta_S(s) = (s', t)$.

Example. Consider a TFSM in Figure 1. If an input *ACK*_3 is applied at state Wait2 at time instances 0, 1, 2 then the TFSM produces the output *ERROR* and moves to state Init. However, if an input *ACK*_3 is applied at state Wait2 at time instance 3 then the TFSM is at state Init and thus, input *ACK*_3 cannot be applied, since a transition under this input is not specified at state Init.

If for each pair $(s, i) \in S \times I$, there is at most one pair $(o, s') \in O \times S$ such that $(s, i, o, s') \in \lambda_S$ then the TFSM is *deterministic*; otherwise, the TFSM is *nondeterministic*. If for each pair $(s, i) \in S \times I$, there is at least one pair $(o, s') \in O \times S$ such that $(s, i, o, s') \in \lambda_S$ then TFSM is *complete*; otherwise,

[1] An abstract output can contain a positive integer for describing a delay of producing an output after an input is applied.

the TFSM is *partial*. The FSM in Figure 1 is partial and nondeterministic.

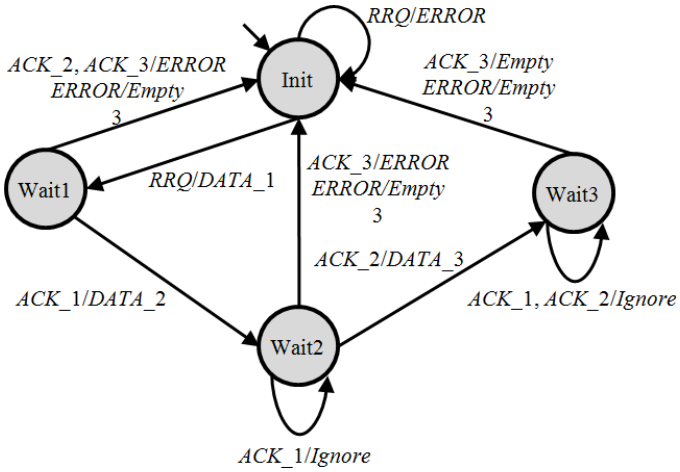

Figure 1. A TFSM for TFTP.

In order to describe the TFSM behavior we use the notion of a timed input that indicates that an input i is applied at time t. Correspondingly a *timed input* is a pair $(i, t) \in I \times Z_+$ where Z_+ is the set of nonnegative integers. In order to extend the transition relation to timed inputs we have to know at which state s' is an FSM when applying an input i at time t and the state s' is determined based on the timeout function [12]. Correspondingly, there is a transition $(s, (i, t), o, s'')$ if and only if there is a transition $(s', i, o, s'') \in \lambda_S$. The behavior of the TFSM is then extended to timed input sequences in a usual way. Consider the TFSM in Figure 1, state Wait2, input *PRQ* and $t = 5$. At time instance 3 the FSM will move to state Init with the timeout ∞ and thus, there is a transition (Wait1, (*PRQ*, 5), *ERROR*, Init) in the TFSM. Given a timed input sequence α, a pair (α, β), $\beta \in O^*$, is a *timed trace* at state s if there exists state s' such that $(s, \alpha, \beta, s') \in \lambda_S$. The set $out_S(s, \alpha)$ is the set of all possible output responses of the TFSM at state s to the timed input sequence α: $\beta \in out_S(s, \alpha)$ iff α/β is a timed trace at state s.

Given a TFSM S, state s of the TFSM is an *input-reachable* state (*ir*-state) if there exists a timed trace (α, β) such that $(s_0, \alpha, \beta, s) \in \lambda_S$ [12]. If a state is not input reachable then it is an *input-unreachable* (*iur*-state). *Iur*-states are not stable: we only can implicitly check that a TFSM has passed this state. The TFSM is *connected* if each state s' is an *ir*-state or there exists an *ir*-state s such that $s' = time_S(s, t)$ for some t. In this paper we consider only connected FSMs. Given a TFSM S, a finite set *TS* of timed input sequences is a *transition tour* of S if for each connected submachine B of S, each *ir*-state b of B and each input i specified at state b, it holds that the set *TS* has a timed input sequence $\alpha.(i, 0)$ such that α takes the TFSM B from the initial state to state b.

Given a transition tour *TS* of the TFSM *S*, the test suite *TS* can detect all output faults at each *ir*-reachable state. Moreover, as we illustrate in the next section, such a test suite can also detect other functional faults. A transition tour of a TFSM can be derived in different ways. A TFSM can be transformed into a classical FSM in such a way that there is the one-to-one correspondence between timed traces of the TFSM and traces of the abstract FSM with the tail input symbol different from 1 [12]. In this case, similar to [4], a special input 1 that corresponds for waiting 1 time unit is added. Since an FSM has to provide an output to each input, we also add a proper output N that corresponds to the case when the FSM does not produce anything. If a timeout at state is more than 1 then we add copies of the next state: the number of copies equals to the value of timeout minus 1. For each other input, there is the same transition at a copy of a state s as at the original state s. If there is an output delay T for a transition i/o then we consider a timed output (o, T) instead of the output o.

In this paper, given a TFSM we derive a test suite *TS* using the following steps.

Step 1. Derive a corresponding classical FSM adding the designated input 1 (*WAIT_1_time_ instance*) and output N.

Step 2. If a derived FSM is nondeterministic derive a set of deterministic submachines in order to cover each transition of the initial FSM.

Step 3. For each deterministic submachine:
- derive a test suite as a transition tour;
- replace each chain $1\ldots1i$ of k transitions $1\ldots1$ and an input i as a timed input (i, k);
- merge all the test suites into a single test suite *TS*.

For each conforming TFTP implementation P and each input sequence $\alpha \in TS$, it holds that $out_P(p_0, \alpha) \subseteq out_S(s_0, \alpha)$. If for some input sequence $\alpha \in TS$, it holds that $out_P(p_0, \alpha) \not\subset out_S(s_0, \alpha)$ then an implementation under test is faulty and the fault is detected by the test suite *TS*.

III. Testing TFTP implementations

The TFTP brief description is taken from [13]. The TFTP (Trivial File Transfer Protocol) is a simple file transfer protocol and it can read and write files (or mail) from/to a remote server. At the first step, there is a request for connection and when the connection is opened the file containing a sequence of blocks of 512 bytes can be sent. If a data packet has less than 512 bytes then there are no packets to be transferred. Each packet contains one data block, and must be acknowledged before the next packet can be sent. A loss of a data packet is prevented by the timeout functions; a lost packet can be retransmitted and the sender has to keep just one packet at hand for retransmission. According to RFC, both connected machines are considered as senders and receivers: one sends data and receives acknowledgments; the other sends acknowledgments and receives data. When an error occurs an error packet is sent. Most errors cause the termination of the connection, and timeouts are used to detect such a termination when the error packet has been lost. Errors are caused by three types of events. The request cannot be satisfied, i.e., the file is not found, there is access violation etc. Another error type occurs when an incorrectly formed packet is received or there is no access to necessary resources. The only error condition that does not cause termination is a situation when the source port of a received packet is incorrect. This protocol is very restrictive, in

order to simplify implementation. Nevertheless, its available implementations still do not conform to the specification.

Five kinds of packets are supported:

- Read request (RRQ);
- Write request (WRQ);
- Data (DATA);
- Acknowledgement (ACK);
- Error (ERROR).

Packages RRQ and WRQ are sent by a client for connection establishment. The connection is established if after sending a packet RRQ (WRQ) the server replies with the first data packet DATA_1 (or after the acknowledgement ACK_0 to zero data packet). Each data packet DATA has the designated number that is an integer started from 1. Correspondingly, when acknowledging the receipt of a data packet a recipient sends the response ACK with the same number. Thus, when the server sends the response ACK_0 as a reply to the request to read the file this only means that the request is confirmed and the server is ready to transfer data. If the server replies with ERROR then the connection is broken and a client has to ask for the connection again. The server uses a special port for requests that is 69 by default. When getting a positive response the server a new port is randomly assigned for data transmission. The latter allows other clients using the main port.

In order to simplify the TFSM that is obtained when describing the protocol behavior we will test only the part that is responsible for getting files from the server (the request RRQ) under the following assumptions. Not more than three packages are transferred as a sequence, no retranslation is allowed and the timeout for waiting a packet equals three seconds. As we show below, even such limitations allow the derivation of a TFSM based test suite that detects faults in known protocol implementations.

Figure 1 represents a TFSM that describes the TFTP behavior when getting the request RRQ. State Init is the initial state. At this state only one input is specified that corresponds to the request to read the file. According to the protocol description, when the first part of DATA (DATA_1) is sent after the request and the FSM enters state Wait1. If there is no such a file or no access to the file, then the server responses with the message about an error (ERROR) and returns to the state Init. The behavior is mostly the same at states Wait1, Wait2, Wait3: the system is waiting for the acknowledgement of the sent data packet. If there is no acknowledgement after 3 seconds then the connection is broken (since due to the above restrictions, the retranslation is not allowed in the simplified TFSM description). Otherwise, the second packet DATA_2 is sent and the system enters state Wait2. If this packet is acknowledged (there is the input ACK_2) then the third packet Data_3 is sent and the system enters state Wait3. If there is the acknowledgement ACK_3 then all the packets have been received and the connection can be closed. Duplicated acknowledgements are ignored by the server.

A transition tour derived for the FSM in Figure 1 has total length of 674 inputs. This has been used for testing some TFTP implementations.

The following implementations being tested:

- class **TFTPServer** defined in the commons-net-2.0.0 library developed by Apache;
- **atftpd** Linux server developed by Jean-Pierre Lefebvre.

Both implementations have been declared to support TFTP [13]. However, the derived FSM with timeouts only partially describes the behavior of the **adftp** server, since there is no parameter in the implementation for changing the number of retranslations. Some mismatching has been detected between the protocol specification [13] and an implementation under test.

When testing the class **TFTPServer** defined in the commons-net-2.0.0 library the following mismatching has been detected. An acknowledgement with the unsent packet number has been ignored while according to the specification [13], these packets have to be declared as ERROR packets and the corresponding message has to be sent.

When testing **atfdp** another mismatching has been detected. When the acknowledgement with any number has been applied the server responded with a DATA packet with the next number. In other words, for example, after applying an input sequence (PRQ,0)(ACK_3,0) the output sequence DATA_1.DATA_4 has been observed which does not match the protocol specification [13].

IV. Conclusions

In this paper, we have studied a test suite that is derived based on the TFSM specification that is extracted from the TFTP description. The test suite has been derived as a transition tour of a corresponding FSM. As the performed experiments show, such test suites can be efficiently used for conformance testing of protocol implementations. A similar approach has been used for testing IRC and SMTP implementations [14, 15] and a number of inconsistencies have been detected. As a future work, we are going to study the fault coverage of test suites which are derived using other 'black box' testing methods.

References

[1] M. P. Vasilevskii, "Failure diagnosis of automata," translated from Kibernetika, No.4, 1973, pp. 98-108.

[2] T. S. Chow, "Test design modeled by finite-state machines," IEEE Trans. SE, vol. 4, No. 3, 1978, pp. 178-187.

[3] M. Dorofeeva, K. El-Fakih, S. Maag, A.R. Cavalli, N. Yevtushenko. FSM-based conformance testing methods: A survey annotated with experimental evaluation. Information and Software Technology, 52, 2010, pp. 1286-1297.

[4] J. Springintveld, F. Vaandrager, P. D'Argenio,.Testing Timed Automata. Theoretical Computer Science, 254(1-2), 2001, pp. 225-257.

[5] A. En-Nouaary, R. Dssouli, F.Khendek. Timed Wp-Method: Testing Real-Time Systems, IEEE TSE 28(11), 2002, pp. 1023-1038.

[6] K. El-Fakih, N. Yevtushenko, H. Fouchal. Testing Timed Finite State Machines with Guaranteed Fault Coverage. TestCom/FATES, 2009, pp. 66-80.

[7] M. G. Merayo, M. Nunez, I. Rodriguez. Extending EFSMs to Specify and Test Timed Systems with Action Durations and Time-outs. IEEE Transactions on Computers, 57(6), 2008, pp. 835-844.

[8] M. G. Merayo, M. Nunez, I. Rodriguez. Formal testing from timed finite state machines // Computer Networks. 52 (2), 2008, pp 432-460.

[9] M. Gromov, D. Popov, N. Yevtushenko. Deriving test suites for timed Finite State Machines. Proceedings of IEEE East-West Design & Test Symposium'08.,2008, pp. 339-343.

[10] M. Gromov, K. El-Fakih, N. Shabaldina, N. Yevtushenko. Distinguishing Non-deterministic Timed Finite State Machines, 11th Formal Methods for Open Object-Based Distributed Systems and 29th Formal Techniques for Networked and Distributed Systems, FMOODS/FORTE, LNCS 5522, 2009, pp. 137-151.

[11] M. Gromov, N. Yevtushenko. Synthesis of distinguishing test cases for timed finite state machines // Programming and Computer Software, 2010. – V. 36. – №.4. – P. 216-224.

[12] M. Zhigulin, N. Yevtushenko, S. Maag, A. Cavalli. FSM-Based Test Derivation Strategies for Systems with Time-Outs // 11th International Conference On Quality Software – Madrid, July 2011. – P. 141-150.

[13] RFC1350 – The TFTP Protocol (revision 2). URL: http://www.rfc-editor.org/rfc/rfc1350.txt

[14] M.V. Zhigulin, A.V. Kolomeez, N.G. Kushik, A.V. Shabaldin. Testing IRC implemintations based on extended finite state machine // Bulletin of the Tomsk Polytechnic University. – 2011. – V. 318, № 5. – P. 81-84 (in Russian).

[15] A. Nikitin, N. Kushik. On EFSM-based Test Derivation Strategies // Proceedings of the 4th Spring/Summer Young Researchers' Colloquium on Software Engineering. – Nizhny Novgorod, Russia. – 2010. – P. 116-119.