An Approach to Evaluating the Performance of Communication Protocols based on Formal Specifications *

Sijian Zhang and Samuel T. Chanson†
Department of Computer Science
University of British Columbia
Vancouver, B.C.
Canada V6T 1Z2

Abstract

This paper proposes an approach to modeling and evaluating the performance of communication protocols based on formal specifications. A combined Extended Finite State Machine (EFSM) and Queueing Network (QN) model is presented which is more sophisticated than either technique alone. Furthermore, the model avoids the state space explosion problem which may arise by the use of reachability analysis *on EFSM. The approach is illustrated using a simple sliding window protocol written in the formal description technique Estelle.*

1 Introduction

Until recently, almost all communication protocols have been specified in natural languages, often resulting in different interpretations by different users. Formal Description Techniques (FDTs) were developed mainly as a means to describing protocols and distributed systems unambiguously. Many FDTs have been developed and used in the past ten years, and three of them are standardized by ISO and/or CCITT. They are Estelle[14], LOTOS[13] and SDL[6].

FDTs provide a basis to i) validate the system design [2], ii) generate test cases for conformance testing [32, 34], iii) simulate system behaviors before implementation [5], iv) implement systems semi-automatically [35], and vi) predict system performance [19, 4, 22, 9, 8].

This paper is concerned with the performance evaluation of communication protocols and systems. There are many ways to model a communicating system. Some of the major techniques are given below:

Finite State Machine (FSM) and **Extended Finite State Machine (EFSM)** are widely used to model communicating processes and systems. This is because most communication protocols are designed as FSMs or EFSMs [33, 15, 16]. Prior work that uses FSMs or EFSMs for performance evaluation can be found in [19, 22, 4, 11].

Queueing model is a classical technique for modeling the behavior of computer systems and analyzing their performance [18, 1]. It is especially useful in modeling resource contentions. The two basic constructs in a queuing network are *queues* and *servers*. The model allows the behavior of tasks in the queues and servers as well as overall system behavior to be estimated. These include queuing time, server utilization and system throughput rate. Typically, some simplifying assumptions about the arrival rates and service rates have to be made.

Petri net is another useful tool to model communicating systems. It has been intensely studied since it was first introduced by C. A. Petri [29] over 30 years ago. A number of modified Petri net models which can be used for performance evaluation have been proposed, including [7].

The **algebraic model** approach originated from Hoare's Communicating Sequential Processes (CSP) [12] and Milner's Calculus of Communicating Systems (CCS) [24]. Timing information can also be integrated with communication algebra to evaluate system performance [28, 31].

Temporal logic was first used to model and reason about concurrent programs [20, 23]. It enhances the traditional logic with some temporal formulae. It is a useful tool to prototype system behaviors related to time [10]. System performance is usually evaluated by using a variant of temporal logic called *interval temporal logic* [27].

EFSM is the most natural technique to describe communication protocols. However, except for the queuing model, the others were initially developed to describe system behaviors and not intended for performance analysis. Therefore, certain features such as time and probabilities have to be added for performance analysis purposes. Examples include timed FSMs (or EFSMs) [4, 22] and timed Petri nets [7, 30, 17, 26]. Both *reachability analyses* with EFSM and Petri Net models suffer from the problem of state space explosion. Some effort in reducing the size of the state space has been made [3], and it continues to be a major research focus.

The approach presented in this paper avoids the state space explosion problem by using a combined

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

†Currently on leave at the Hong Kong University of Science and Technology.

0-8186-3670-X/93 $03.00

EFSM and Queueing Network model. The main idea in our approach is to map the communication protocols specified in a formal description technique into EFSM (actually a variant designed for performance analysis called PEFSM), and use the PEFSM model to compute the system parameters required by the Queuing Network model. Queueing theory is then applied to estimate the system performance. Thus, dynamic resource contentions are also taken into account which solves the major weakness of using EFSM alone.

The rest of the paper is organized as follows. Section 2 describes the model approach. Section 3 introduces the methodology to evaluate performance based on the proposed model. Section 4 and Appendix A provide an example a simple protocol specified in Estelle as a case study. Finally, Section 5 concludes the paper.

2 Model description

We assume the specification is written in Estelle [14]. For simplicity, we further assume the specification consists of one module only. This is not a severe limitation since a multi-module Estelle specification can be transformed into a single module specification [32].

Our model consists of a variant of EFSM, which shall be called PEFSM, and a queuing network (QN). Both the PEFSM and QN are to be derived from the Estelle specification.

2.1 PEFSM

PEFSM is similar to EFSM in that both are based on states and transitions. The major differences are in the addition of some performance aspects such as transition probabilities and features specific to communication systems such as channels, and more refined input and output time models.

Formally, the PEFSM is defined as

$$M = (Q, I, O, C, \Delta, q_0)$$

where Q – the set of the states in the protocol or communicating system
I – the set of all possible input messages
O – the set of all possible output messages
C – the set of all channels used by the incoming and outgoing messages
$\Delta : Q \times I \times O^* \rightarrow Q$
q_0 : idle state

Δ is the set of state transitions. Each transition specifies an input and zero, one or more outputs accompanied with a state change.

2.1.1 Transition time

Unlike the case of EFSM, each transition in the PEFSM consists of one incoming message and zero, one or more outgoing messages. Therefore, conceptually, the transition execution time can be divided into two parts : the time for the incoming message (I) and the time for the outgoing message(s) (o1,o2,...), as shown in Figure 1.

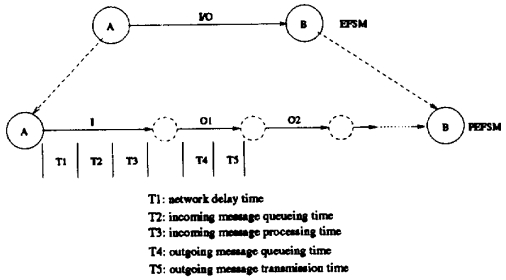

Figure 1: Time model for a transition

In our model, the time for an incoming message consists of the *network delay time*, the *queuing time* and *processing time* (see below); the time for an outgoing message consists of *queuing time* and *transmission time*. The *processing time* has the same meaning as in [4, 22] and includes all the CPU times to process the incoming message and execute the transition. Its value can be obtained from computation complexity analysis, measurement, or assigned by the evaluator. *Transition time* includes the *network delay time* and *queueing times*. In this paper, the *network delay time* is not taken into account because only one module is considered.

We use a vector, called M_e, to describe all the transition processing times:

$$M_e = (f_1, f_2, ..., f_n)$$

where $n = |\Delta|$ (i.e. the number of transitions in the PEFSM); f_i (i=1,...,n) is the CPU processing time needed in transition i. It may be expressed as a function or a constant.

A matrix, called M_o, is used to specify the number of each class of outgoing messages generated by each transition :

$$M_o = \begin{pmatrix} f_{11} & f_{12} & \cdots & f_{1m} \\ \vdots & & & \vdots \\ f_{n1} & f_{n2} & \cdots & f_{nm} \end{pmatrix}$$

where n has the same meaning as before; $m = |O|$ (i.e. the number of outgoing message classes); f_{ij} is the number of type j outgoing messages generated by transition i. It may be expressed as a function or a constant.

M_o can be derived from the formal specification.

2.2 Queuing Network (QN)

Since the EFSM models abstract system states rather than explicit resources, it is difficult to describe resource contention behavior. For this purpose, we use a queuing network model.

Our queuing network is constructed with regard to a single module without any embedded submodule in it. The queuing network consists of 6 components as shown in Figure 2 :

1. queue(s) for incoming messages
   There may be more than one queue for each channel to accept incoming messages.

2. service center(s) for transition execution

   For each module, conceptually, there is only one service center to execute the fired transitions.

3. queue(s) for outgoing messages

   Each channel has one queue to store the outgoing messages for transmission. More than one channel may share a queue (cf. Estelle's *common queue*).

4. service center(s) for transmission (i.e., channel(s))

   Each outgoing message is transmitted through its specific channel with respect to a transition.

5. input-to-output amplifier

   An incoming message may generate zero, one or more outgoing messages. We provide *amplifiers* to depict this.

This queuing network model is somewhat different from the conventional queuing model where service centers typically correspond to hardware resources. When modeling the system at the specification level, the service center for transition execution in a module is a *conceptual CPU*, not a physical one. Possibly, more than one module in an Estelle specification will be implemented on the same computer system which has only one CPU. A **mapping** between *specification level* and *hardware level* is needed which is similar to the one between *software level* and *hardware level* introduced in [21]. It is not difficult to see that both the PEFSM and the QN can be easily derived from an Estelle specification. An example is given in Section 4.

2.3 Work-load model

The *work load* of a computer system is defined as the set of all inputs (i.e., programs, data, and commands) the system receives from its environment. From the formal specification point of view, the work load is a set of external variables which influence system performance. It describes the resource demands of jobs or some characteristics of jobs from which the resource demands can be derived. The workload model must be compatible to the underlying system models which, in our case, are the PEFSM and the Queuing Network model.

Transition times and transition probabilities are commonly used as work-load parameters in performance evaluation studies based on EFSM [22, 4, 8, 3]. They are also used in our PEFSM model. Moreover,

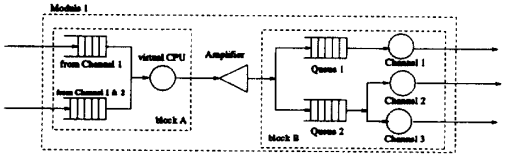

Figure 2: An example of a queuing model from an Estelle specification

additional information on the transition times derived from the specification is included.

As mentioned earlier, the transition time is composed of two parts : the time for CPU processing and the time for output transmission. These two times can be computed from the vector M_e and the matrix M_o. M_e and M_o can be derived from a set of external variables in the specification using data flow analysis [32]. We denote this set of external variables as the W set.

Similar to traditional queuing network models for performance evaluations [21, 1], the arrival rates of incoming messages and the average service demands for each class of incoming and outgoing messages are included in our work load model. However, the service demands are not given directly in the form of constants or probability distributions. Instead, they are computed from M_e and M_o as shown in the next section.

3 Performance evaluation

First, some of the notations used in the rest of the paper are explained here :

- p''_{imj} is the probability of the system going from state i to state j when a class m message is received. This can be used to model non-determinism.
- incoming message arrival rates $\vec{\lambda_I}$

  Assuming there are n classes of incoming messages,

  $$\vec{\lambda_I} = (\lambda_{I_1}, \lambda_{I_2}, \cdots \lambda_{I_n})$$

In the following subsections, the algorithms for performance evaluation are presented.

3.1 Computation of state-message probabilities P_{sm}

An element p_{ij} of the matrix denotes the probability the system at state i will receive a class j message. Note that the system may not be able to receive some message classes in certain states.

Input: $\vec{\lambda_I}$
Output: P_{sm}
Algorithm:
For (each state s in the PEFSM of the specification) do :

1. Let K_s be the set of message classes that can be received at state s, then for each $j \in K_s$

   $$p_{sj} = \frac{\lambda_{I_j}}{\sum_{i \in K_s} \lambda_{I_i}}$$

   where p_{sj} is an element of matrix P_{sm}.
2. For all other j (i.e. $j \notin K_s$), set

   $p_{sj} = 0$

   Endfor;

Endfor.

3.2 Computation of next-state probabilities P_{ss}

This is given as a matrix. An element p'_{ij} in the matrix P_{ss} represents the probability that the system will go from state i to state j.

Input: P_{sm}
Output: P_{ss}
Algorithm:
Let S be the total number of states in the system.
For i=1 to S do
For j=1 to S do
1) If there is no transition from state i to state j, then $p'_{ij} = 0$
2) else $p'_{ij} = \sum_{m \in M} p_{im} \times p''_{imj}$ where p_{im} is an element of P_{sm}; M is the set of message classes that go from state i to state j; p''_{imj} is the probability the system will go from state i to state j when a class m message is received.
Endfor;
Endfor.

3.3 Computation of steady-state probabilities P_s

If the PEFSM for a communication protocol is *aperiodic* and *irreducible*, the steady-state probabilities are unique and computable [25].

Given the next-state probabilities P_{ss}, the steady-state probabilities P_s can be computed by solving the following matrix equation :

$$P_s = P_{ss} \cdot P_s$$

If the PEFSM is periodic, the steady-state probability of every state is $\frac{1}{n}$, where n is the number of states in the system.

3.4 Computation of transition probabilities Q_t

The probability that a transition taken is transition t is denoted as q_t. Q_t is a vector consisting of the transition probabilities for all the transitions in the system, i.e., all the $q_t s$.

It is determined by the steady-state probability of the current state s (the start state), the class m message arrival probability in this state, and the probability of taking transition t in these circumstances. i.e.,

$$q_t = p_s \cdot p_{sm} \cdot p''_{smj} \quad (1)$$

where p_s is an element of P_s and s is the start state of transition t; p'_{sm} is an element of P_{sm} and represents the probability of receiving a class m message at state s; p''_{smj} is the probability of the system taking transition t which corresponds to going from state s to state j when a class m message is received.

3.5 Computation of average service demand D_m of class m incoming messages

The service demand of an incoming message is the time needed to execute the transition invoked by the message. A message of a given class may invoke different transitions at different states. This situation is taken into consideration when computing the average service demand for class m messages :

$$D_m = \sum_t q_t \cdot f_t \quad (2)$$

where transition t is fired because of the arrival of a class m message; q_t is the probability of transition t (see (1)) and f_t is the execution time of transition t, which is an element of the matrix M_e.

3.6 Computation of the generation rates of outgoing messages

First, the mean amplifier coefficient matrix M_{IO} is computed. The matrix describes the number of outgoing messages which will be provoked by each class of incoming messages.

Let M_{mt} be the matrix of transition probabilities under steady-state situation with respect to each class of incoming message. m_{ij} is an element of the matrix and represents the probability of a class i message firing transition j.

If a class i incoming message fires transition j, then $m_{ij} = q_j$ where q_j is an element of Q_t; otherwise, $m_{ij} = 0$. We normalize each row of the matrix M_{mt}, i.e. $\sum_{j=1}^{n} m_{ij} = 1$ ($n = |\Delta|$, the total number of the transitions).

Then, we compute

$$M_{IO} = M_{mt} \times M_o \quad (3)$$

The generation rates for outgoing messages $\vec{\lambda_O}$ is given by

$$\vec{\lambda_O} = \vec{\lambda_I} M_{IO} \quad (4)$$

where $\vec{\lambda_I}$ is the rates of incoming messages.

3.7 Evaluating performance

System performance can now be computed using queuing analysis techniques on the parameters described in Sections 3.1 to 3.6. First, the queuing time for the transition execution service center is computed.

Consider Block A of Figure 2. We assume that the queuing discipline is *first in first out* (FIFO) and messages have equal priority. The queues from the different channels are aggregated into one queue, but the different classes of messages still retain their distinct identities. Therefore, the model is reduced to a simple multi-job class single server queuing model.

Using the *open model solution technique* [21], we have

$$U = \sum_{i=1}^{n} \lambda_{I_i} D_i; \quad (5)$$

$$R_c = \frac{D_c}{(1 - \sum_{i=1}^{n} \lambda_{I_i} D_i)}; \quad (6)$$

$$T_c = R_c - D_c; \quad (7)$$

and

$$L = \sum_{c=1}^{I} \lambda_{I_c} T_c \quad (8)$$

where
$n = |\lambda_I|$;
U is the utilization of the CPU in Block A;
R_c is the mean residence time for class c messages;
T_c is the mean queuing time for class c messages;
L is the mean queue length.

Next, consider Block B. The mean queuing time for outgoing message channel k (k=1,2) is calculated as follow:

Case 1: Channel k has its own dedicated queue i.e., individual queue in Estelle (e.g., Channel 1 and Queue 1 in Block B of Figure 2).

Assume that

- λ_{O_k} is the message arrival rate to Channel k.
- B_k is the bandwidth of Channel k;
- $\overline{s_k}$ is the average message size handled by Channel k;

λ_{O_k} is obtained from Equation (4); B_k and $\overline{s_k}$ are to be provided by the evaluator.

The average transmission time is : $D_k = B_k \times \overline{s_k}$.

Using the *open model solution technique* [21], we get the mean residence time

$$R_k = \frac{D_k}{1 - \lambda_k D_k}$$

The mean queuing time is therefore : $T_k = R_k - D_k$.

By Little's law, the mean queue length is : $L_k = \lambda_k T_k$.

Case 2: Queue r is the shared buffer for Channels 1 through C, i.e., common queue in Estelle (e.g., Queue 2 in block B of Figure 2).

Assume that

- $\lambda_1, \lambda_2, ..., \lambda_c$ are the message arrival rates to Channels 1, 2, ... and c, respectively.
- $D_1, D_2, ..., D_c$ are the service demands of message classes 1, 2, ... and c, respectively.

This common-queue model is neither a traditional G/G/c model nor a simple open multi-class model. It is a combination of the two. This is because the messages in the queue belong to different classes and each will go to its own specific service center (i.e. channel in our model) for service only.

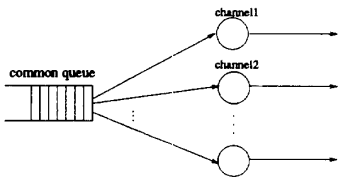

Figure 3: A common queue model for multi-channels

We will use the *open multi-job class model solution technique* [21, 1] to compute an approximate solution for this common-queue model.

The throughput for each channel i is simply : $H_i = \lambda_i$.

The utilization of channel i is : $U_i = \lambda_i D_i$.

The residence time of class i messages is : $R_i = \frac{D_i}{(1 - \sum_{k=1}^{c} U_i)}$.

The queue length of class i messages is : $L_i = \lambda_i R_i$.

The total queue length in the common queue is : $L = \sum_{k=1}^{c} L_k$.

By Little's law, the queuing time for a message in the common queue is :

$$T_q = \frac{L}{\sum_k^c \lambda_k}.$$

4 A case study: Performance prediction of sliding window protocol

We now illustrate the ideas presented in the previous sections using a simple *sliding window protocol*. The Estelle specification of part of the protocol is given in Appendix A. The module structure of the specification is given in Figure 4. We study the performance of the *transmitter* module whose queuing network model is given in Figure 5.

From the specification, there are five transitions T1, T2, T3, T4 and T5 in the transmitter module. The time complexity for these transitions is given by

$$M_e = (f_1, f_2, f_3, f_4, f_5)$$

where

$$f_1() = c_1$$
$$f_2(Lowest\_Unacked, PDU.seq) = (PDU.seq - Lowest\_Unacked)c_2 + c_3$$
$$f_3() = c_4$$
$$f_4() = c_5$$
$$f_5() = c_6$$

and $c_1, c_2, c_3, c_4, c_5, c_6$ are constants which are system dependent.

There are a total of 4 classes of outgoing messages produced by the transmitter. They are *U.Data_indication, CT.DT, T.timer_request and T.timer_cancel*. The number of each message class generated by each transition is given in the matrix M_o.

$$M_o = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & m1 \\ 0 & 0 & 0 & 0 \\ 0 & m2 & m2 & m2 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

where

$$m1 = (PDU.seq - Lowest\_Unacked)$$
$$m2 = (Highest\_Sent - seq)$$

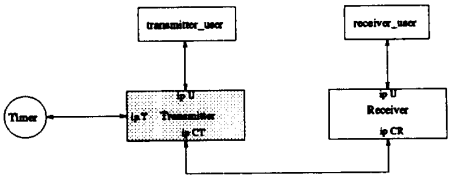

Figure 4: Module relation graph of sliding window protocol

M_e and M_o are derived from the specification. 'Lowest_Unacked' and 'Highest_Sent', 'PDU.seq' and 'seq' are parameters or variables in the specification. 'Lowest_Unacked' and 'Highest_Sent' are the lower edge and the upper edge of the transmitter window. 'PDU.seq' is the sequence number of the message received. 'seq' is the sequence number of the message whose timeout clock has expired while waiting for acknowledgement.

Using data flow analysis, we have

$$W = \{seq, PDU.seq, Highest\_Sent\},$$

The parameters in set W are assumed to have mean values given by

$$\overline{f_2} = (TWS/2)\, c_2 + c_3$$
$$\overline{f_{24}} = TWS/2$$
$$\overline{f_{42}} = TWS/2$$
$$\overline{f_{43}} = TWS/2$$
$$\overline{f_{44}} = TWS/2$$

where TWS is the window size.

Since there is no nondeterministic choice in the specification, i.e., the system's state change can be uniquely determined by the current state and the incoming message, we have $p''_{imj} = 1$ (see section 3).

In this protocol, there are five classes of incoming messages corresponding to the five transitions:

1. U.Data_Request
2. CT.AK where $(PDU.seq \geq Lowest\_Unacked)$ and $(PDU.seq \leq Highest\_Sent)$
3. CT.AK where $(PDU.seq < Lowest\_Unacked)$ or $(PDU.seq > Highest\_Sent)$
4. T.timer_response
   where $(seq \geq Lowest\_Unacked)$ and $(PDU.seq \leq Highest\_Sent)$
5. T.timer_response
   where $(seq < Lowest\_Unacked)$ or $(PDU.seq > Highest\_Sent)$

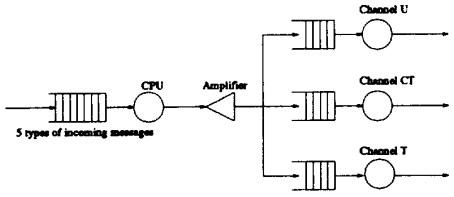

Figure 5: The queuing network model for *transmitter*

Let

$$\vec{\lambda_I} = (\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5)$$

and

$$\vec{D} = (D_1, D_2, D_3, D_4, D_5)$$

denote the arrival rates and the mean service demands for CPU of the five classes of incoming messages, respectively.

Suppose that the following work load parameter values are known:

- $\vec{\lambda_I} = (\lambda_1, 100, 100, 50, 10)$
  where the unit here is messages/second.
- $c_1 = c_4 = c_5 = c_6 = 0.0001$ and $c_2 = c_3 = 0.0004$
  where the unit here is second.
- mean transmission times for outgoing messages are 0.001 second.
- The window size of the transmitter is 2.

In each of the following figures, the performance results are obtained using the analytic methods presented in Section 3 and also by simulation. The simulation results are shown using the darker solid lines. As can be observed, the differences between analytic and simulation results are acceptably small, which validated the analytic model.

The utilization and the queue length of the CPU for the transmitter vs the user message arrival rates are given in Figures 6 and 7 respectively.

The utilization and the queue length of channel CT vs the user message arrival rates are given in Figures 8 and 9 respectively.

The mean system response time vs the user message arrival rate is given in Figure 10. System response time is defined as the period between the time the transmitter receives a message and the time the corresponding message leaves channel CT.

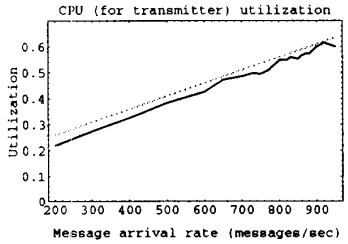

Figure 6: CPU utilization

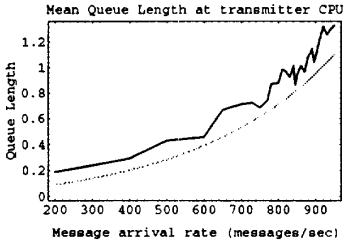

Figure 7: Queue length at CPU

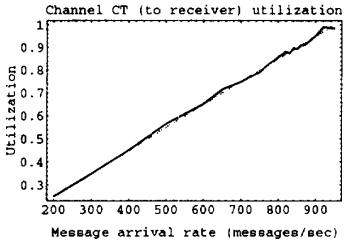

Figure 8: Channel CT utilization

5 Summary and Conclusions

We have presented a model for the performance evaluation of communication protocols and systems from their formal specifications. The model is based on the Queuing Network Model and a variant of the Extended Finite State Machine called PEFSM. This approach combines the advantages of both techniques. The formal specification is mapped onto the PEFSM model in a straightforward manner. This also allows a more detailed description of the system and workload than is the case for straight queuing network models. The main purpose of the PEFSM model is to derive input parameter values for the queuing network model from which system performance is calculated. In this way, dynamic resource contention is modeled and state space explosion, which is common in traditional reachability analysis on EFSM, is avoided. Simulation results indicate this approach can achieve acceptable accuracy.

References

[1] Arnold O. Allen. *Probability, Statistics, and Queueing Theory with Computer Science Applications.* Academic Press, Inc., second edition edition, 1990.

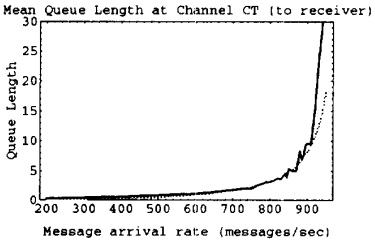

Figure 9: Queue length at Channel CT

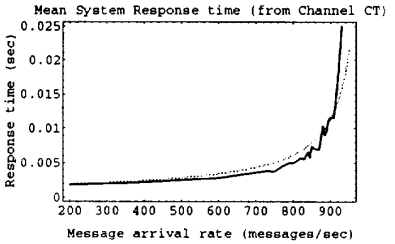

Figure 10: Mean system eesponse time

[2] P. Azema, G. Papapanagiotakis J. C. Lloret, and F. Vernadat. Estelle Validation and PROLOG Interpreted Petri Nets. In *FORTE'89*, 1989.

[3] 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.

[4] 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).

[5] S. Budkowski. Estelle development toolset (EDT). *Computer Networks and ISDN Systems*, 25(1):63–82, 8 1992.

[6] CCITT. *Specification and Description Language – Recommendation Z.100.* Geneva, Switzerland: CCITT press, 1986.

[7] Michel Diaz. Modelling and analysis of communication and cooperation protocol using Petri Net based models. In *Protocol Specification, Testing, and Verification, IFIP.* Elsevier Science Publishers B. V. (North-Holland), 1982.

[8] D. Fernandez, E. Vazquez, and J. Vinyes. Io: An Estelle Simulator for Performance Evaluation. In *FORTE'91*, 1991.

[9] 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).

[10] Roger Hale. Using Temporal Logic for Prototyping: The Design of a Lift Controller. In H.S.M. Zedan, editor, *Real-Time Systems, Theory and Appliactions*, pages 81–118. Elsevier Science Publisher B.V. (North-Holland), 1990.

[11] E. Heck, D. Hogrefe, and B. Muller-Clostermann. Hierarchical Performance Evaluation Based on Formally Specified Communication Protocols. *IEEE Transactions on Computers*, 40(4), 4 1991.

[12] Charles A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International, 1985.

[13] 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.

[14] 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.

[15] ISO TC97/Sc16 N2576. *Formal specification of Transport protocol in Estelle.* ISO, 1986.

[16] ISO/TC 97, Information processing systems. *Information processing systems - Data communication - High-level data link control procedures - Discription of the of the X.25 LAPB-compatible DTE data link procedures.* ISO 7776 - 1986. ISO, first edition, 12 1986.

[17] J-L. Roux and G. Juanole. Functional and Performance Analysis using Extended Time Petri Nets. In *International Workshop on Petri Nets and Performance Models.* The Computer Society of the IEEE, 1987.

[18] L. Kleinrock. *Queueing Systems: Vol. 1, Theory and Vol. 2, Computer Applications.* John Wiley and Sons, 1975.

[19] 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).

[20] Leslie Lamport. The Temporal Logic of Actions. Technical Report 79, Digital Systems Research Center, 130 Lytton Avenue, Palo Alto, California 94301, 12 1991.

[21] E.D. Lazowska, G.S. Graham J. Zahorjan, and K.C. Sevcik. *Quantitative System Performance - Computer System Analsysis Using Queueing Network Models.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1984.

[22] 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).

[23] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems.* Springer-Verlag, 1992.

[24] R. Milner. *A Calculus of Communicating Systems.* Springer Verlag, 1980.

[25] Michael K. Molloy. *Fundamentals of Performance Modeling.* Macmillan Publishing Company, 1989.

[26] Michael K. Molloy. Petri Net Modelling - The Past, the Present, and the Future. In *Petri Nets and Performance Models : The Proc. of the 3rd Intl. Workshop (PN&PM 89 - Kyoto, Japan).* IEEE, 1989.

[27] Ben Moszkowski. A Temporal Logic for Multilevel Reasoning about Hardware. *IEEE Computer,* 18(2), 1985.

[28] Nihal Nounou and Yechiam Yemini. Algebraic Specification-Based Performance Analysis of Communication Protocols. In Y. Yemini, R. Strom, and S. Yemini, editors, *Protocol Specification, Testing and Verification IV (IFIP).* Elsevier Science Publishers B. V. (North-Holland), 1984.

[29] C. A. Petri. Kommunikation mit Automaten. Technical report, Schriften des Rheinisch-Westfalischen Institutes fur Instrumentelle Mathematik an der Universitat Bonn, Heft 2, Bonn, W. Germany, 1962. translation: C. F. Greene, Supplement 1 to Tech. Rep. RADC-TR-65-337, Vol. 1, Rome Air Development Center, Grifiss Air Force Base, N.Y., 1965.

[30] R. R. Razouk and C. V. Phelps. Performance Analysis Using Timed Petri net. In *Protocol Specification, Testing, and Verification, IV (IFIP),* pages 561–576. Elsevier Science Publishers B. V. (North-Holland), 1985.

[31] Nathalie Rico and G.v. Bochmann. Performance description and analysis for distributed systems using a variant of LOTOS. In *11th Intnl. IFIP WG6.1 Symposium on Protocol Specification, Testing, and Verification.* Elsevier Science Publishers B. V. (North-Holland), 1991.

[32] B. Sarikaya, G.v. Bochmann, and E. Cerny. A Test Design Methodology for Protocol Testing. *IEEE Transaction on Software Engineering,* SE-13(5), 5 1987.

[33] Andrew S. Tanenbaum. *Computer Networks.* Pretice-Hall, Inc., 1988.

[34] H. Ural. A test derivation method for protocol conformance testing. In *Proc. of 7th IFIP/WG6.1 International Workshop on Protocol Sepcification, Testing, and Verification,* Zurich, Switzerland, 5 1987.

[35] J.F. Lenotre V. Chari and E. Mariani. The Estelle translator. In *The Formal Description Technique Estelle,* pages 325–351. North-Holland, Amersterdam, 1989.

Appendix A: An Estelle specification for sliding window protocol

In this appendix, an Estelle specification example is included which specifies the *sliding window protocol*. However, only the *transmitter* module of the specification is given below due to size limitation.

```
specification Sliding_Window_Protocol;
  ......
 module transmitter_head systemprocess;
  ip U  : user_chan(provider);
     CT : comm_chan(DT_sender);
     T  : timer_chan(user);
 end;
 body transmitter for transmitter_head;
  const transmitter_window_size =
                  any integer;
  state SENDING;
  { save data until acknowledgment }
  procedure buf_save(s : seq_type;
        d : user_data_type); primitive;
  { free data after acknowledgment }
  procedure buf_free(s:seq_type); primitive;
  { retrieve user data from buffer}
  function  buf_retrieve(s : seq_type) :
                  user_data_type; primitive;
  { construct a DT PDU from the user data
                  and sequence number }
  function  PDU_DT(s : seq_type; d :
      user_data_type):DTPDU_type; primitive;
  var
   Lowest_Unacked, Highest_Sent : seq_type;
   TWS : integer;
  initialize
   to SENDING
    begin
     Lowest_Unacked := 1;
     Highest_Sent := 0;
     TWS := transmitter_window_size;
    end;
  trans
T1: from SENDING to same
    { transmit while window not full }
     when U.Data_Request
      provided Highest_Sent-Lowest_Unacked<TWS
       begin
        Highest_Sent:=Highest_Sent+1;
        output T.timer_request(Highest_Sent);
        output CT.DT(PDU_DT(Highest_Sent, data));
        buf_save(Highest_Sent, data);
       end;
T2: from SENDING to same
    { receive acknowledgment }
     when CT.AK
      provided (PDU.seq >= Lowest_Unacked)
                and (PDU.seq <= Highest_Sent)
       var s : seq_type;
       begin
        for s:=Lowest_Unacked to PDU.seq do
         begin
          output T.timer_cancel(s);
          buf_free(s);
         end;
        Lowest_Unacked := PDU.seq + 1;
       end;
T3:   provided otherwise
      { receive ack not in window }
       begin  { ignore this ack }  end;
T4: from SENDING to same { Timer response }
     when T.timer_response
      provided (seq >= Lowest_Unacked) and
                (seq <= Highest_Sent)
       var s : seq_type;
       begin
        for s:=seq to Highest_Sent do
         begin
          output T.timer_cancel(s);
          output CT.DT(PDU_DT(s,
                   buf_retrieve(s)));
          output T.timer_request(s);
         end;
       end;
T5:   provided otherwise
       begin
        {ignore the response for sequence number
         outside the window. This can happen
         when an AK arrives just when a timer
         response occurs.}
       end;
  end; { transmitter body }
  ......
  modvar transmitter_instance : transmitter_head;
         receiver_instance : receiver_head;
         transmitter_user_instance,
         receiver_user_instance : user_head;
         timer_instance : timer_head;
  initialize
  begin
    init transmitter_user_instance with
                transmitter_user;
    init receiver_user_instance with receiver_user;
    init transmitter_instance with transmitter;
    init receiver_instance with receiver;
    init timer_instance  with timer;
    connect transmitter_user_instance.U to
                transmitter_instance.U;
    connect receiver_user_instance.U to
                receiver_instance.U;
    connect transmitter_instance.CT to
                cms_instance.CR;
    connect transmitter_instance.T to
                timer_instance.T;
  end;
end.
```