Application of symbolic FSM Markovian analysis to protocol verification

M.Baldi, A.Macii, E.Macii and M.Poncino

Abstract: Computer-aided design tools for Markovian analysis and verification of digital circuits have gained much interest in the last few years. This is mainly due to the advent of effective data structures, e.g., binary and algebraic decision diagrams, for Boolean and pseudo-Boolean function representation and manipulation.The authors illustrate how those tools can be successfully exploited to analyse the behavior and verify the correctness of a communication protocol.They first consider the case of single protocol entities running in isolation, and present a simple application example (i.e. the sender entity of the alternating bit protocol). Then, informally illustrate how the analysis approach can be extended to the general case of multi-layer protocol stacks and complete communication systems.

1 Introduction

Different methods have been proposed and used so far to analyse the behavior and verify the correctness of communication protocols. The most common is simulation. Although this approach is extremely valuable for understanding what a model does in selected cases, it is inadequate to guarantee that a given property (or behavior) holds for all possible cases. To perform such “exhaustive” analysis and verification, other solutions are required. In view of this probabilistic techniques have attracted a lot of interest in recent times, thanks to the promising results they have provided in other fields of electrical engineering and computer science.

A communication protocol usually represents a system whose behavior can be specified through a finite state machine (FSM). It is known that the probabilistic (or Markovian) behavior of a FSM can be investigated by regarding its transition structure as a Markov chain [1, 2]. As a consequence, studying the Markov chain is related to performing reachability analysis on a FSM. (Reachability analysis, also called state space traversal, consists of determining the set of states of the FSM that can be reached from a given initial state after any sequence of vectors of any length is applied to the inputs of the FSM.) Finite state machines are often used to model digital systems in the context of logic synthesis and formal hardware verification. Therefore, a well-established technology for analyzing large FSMs (i.e., those corresponding to complex circuits) is currently available. More specifically, exact and approximate procedures to quickly, yet effectively, carry out the state space traversal, as well as the Markovian analysis of a given FSM, do exist. All these procedures rely on the capability of compactly representing Boolean and pseudo-Boolean (i.e., real-valued) functions provided by symbolic data structures such as binary decision diagrams (BDDs) [3] and their extensions (e.g., multi-terminal binary decision diagrams (MTBDDs) [4] and algebraic decision diagrams (ADDs) [5]).

In this paper we illustrate how symbolic FSM Markovian analysis can be exploited to perform quantitative investigation and verification of properties of communication protocols. We first consider the simple case of isolated protocol entities. Then, we informally discuss how the modularity of the approach makes it applicable to entire, multi-layer protocol stacks and complete communication systems.

Protocol analysis and verification based on Markovian investigation of the finite state structure that describes its behavior has been in use for some time. However, a serious constraint on the applicability of this procedure was posed by its high computational complexity. In practice, only structures with at most a few tens of states could be analyzed. This major limitation is removed by the use of symbolic calculations. Much larger systems (billions of states and beyond) can now be handled; therefore, the Markovian analysis and verification paradigm becomes applicable in practical and realistic situations, as already demonstrated by applications such as probabilistic model checking and property verification [6, 7].

2 Background

2.1 Boolean functions and BDDs

An n-input, m-output Boolean function F is a mapping from an n-dimensional Boolean space to an m-dimensional Boolean space, F: $B^n \mapsto B^m$, where $B = \{0, 1\}$. B^n is called the *domain* of F, and B^m is called the *co-domain* (or *image*) of F. If $m > 1$, then F is a *multiple-output* function. If $m = 1$, then F is a *single output* function, and we denote it with f.

The *support* of a Boolean function is the set of variables the function depends on.

Given an n-input, single-output Boolean function, $f(x_1, \ldots, x_n)$, the positive and negative cofactors of f, with respect to variable x_i, are defined as:

$$f_{x_i} = f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n)$$

© IEE, 1999
IEE Proceedings online no. 19990645
DOI: 10.1049/ip-cdt:19990645
Paper first received 8th June 1998 and in revised form 21st May 1999

The authors are with the Politecnico di Torino, Dipartimento di Automatica e Informatica, Torino, Italy 10129

and

$$f_{x_i'} = f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$$

Among the various data structures available for the representation of Boolean functions (e.g., truth tables, Karnaugh maps, algebraic expressions), the binary decision diagrams (BDDs) [3] have emerged as the most appropriate for applications such as automatic synthesis, verification, simulation, and testing of complex VLSI systems.

A BDD is a directed acyclic graph (DAG); the root node of the DAG identifies the function, f, represented by the BDD, the internal nodes are labeled with the variables belonging to the true support of f (i.e., the set of variables on which f actually depends), and the terminal nodes are labeled with the values 0 and 1.

The BDDs we consider are reduced, that is, they do not contain duplicated and redundant nodes. In addition, they are ordered, that is, all the variables appear in the same order along all paths from the root to the terminal nodes.

2.2 Pseudo-boolean functions and ADDs

An n-input pseudo-Boolean function, $f{:}B^n \rightarrow S$, is a mapping from an n-dimensional Boolean space to a finite set of elements S. Different data-structures have been proposed for storing and manipulating functions of this type. In this work, we use the algebraic decision diagrams (ADDs) [5].

The most important operators for efficient manipulation of the ADDs are: ITE, APPLY, and ABSTRACT. ITE takes three arguments: f, an ADD restricted to have only 0 or 1 as terminal values, and g and h, generic ADDs. It is defined by:

$$\text{ITE}(f, g, h) = f \cdot g + f' \cdot h$$

APPLY takes one operator, op (e.g., $+$, $-$, $\times$), and two operand ADDs as arguments; it applies op to all corresponding elements of the two operands and returns the resulting ADD.

ABSTRACT reduces the dimensionality of its argument function through existential arithmetic abstraction of some variables. Let u be the support of a pseudo-Boolean function $f(u)$, and let x and y be two sub-sets of u such that $x \cup y = u$. The arithmetic existential abstraction of x from $f(u)$ with respect to the arithmetic sum is defined as:

$$\backslash_x^+ f(u) = \sum_x f(u)$$

This definition tells that the ABSTRACT operator computes the arithmetic sum of all the cofactors associated with the minterms of the x-variables and thus originates a function that depends only on the y variables.

2.3 Finite state machines (FSMs)

A Mealy-type finite state machine (FSM), M, is defined as the 6-tuple:

$$M = (X, Z, S, s^0, \Delta, \Lambda)$$

where X is the input alphabet, Z is the output alphabet, S is the finite set of states, $s^0 \in S$ is the unique reset state, $\Delta{:}X \times S \mapsto S$ is the next-state function, and $\Lambda{:}X \times S \mapsto Z$ is the output function. An FSM, M, can also be represented by a state transition graph (STG). Every vertex of such a graph corresponds to a state of M, and is labeled with an element of S, while every edge corresponds to a transition, and is labeled with an element of $X \times Z$. States that can be reached, under some input sequences, from the reset state are called *reachable* states.

In order to write functions Δ and Λ as (possibly multiple-output) Boolean functions, the symbols of the input and output alphabets must be encoded using distinct sets of binary variables. In addition, the symbolic states in S are assigned unique binary codes and they are represented using an appropriate set of binary variables.

3 Symbolic FSM analysis techniques

3.1 FSM traversal

The FSM traversal problem consists of computing the set of reachable states of a given FSM starting from a set of initial states. Traditional approaches, based on the explicit representation of the STG, are limited because the entire graph can not be stored unless it is for a small machine. A new approach based on breadth first search (BFS) and called *symbolic FSM traversal*, has been originally proposed by Coudert *et al.* [8], and refined in [9–12].

The key ideas of the method are the use of the characteristic functions to represent sets of states reached at any point during the BFS traversal, and the use of BDDs to represent these characteristic functions. The complexity of FSM traversal is not directly related to the number of states of the machine, and FSMs with billions of states have been traversed successfully using this method. In fact, the explicit STG is no longer needed: the algorithm directly manipulates the next state function of the machine to be traversed.

In general, the technique is very efficient because states are manipulated implicitly using their characteristic functions and not one by one as previous algorithms did.

A simplified description of the BFS traversal procedure is presented in Fig. 1. The key operation is *image computation*. Given a multiple-output Boolean function, $F{:}B^n \mapsto B^m$, and a subset c of the domain, the image of c under F is defined by:

$$\text{IMAGE}(F, c) = \{F(x) | x \in c\}$$

Initially $from^0 = S^0$ is the characteristic function of the set of initial states of the machine. The characteristic function *reached* represents the set of states that have been reached so far from the initial states. Some states in to^i may have been reached previously, so a set difference operation with *reached* is needed in order to compute new^i, the states reached in this iteration for the first time. The set is used to check the termination condition and is accumulated in *reached*. The procedure is guaranteed to terminate in a finite number of steps, because the number of states is finite and *reached* is non-decreasing.

```
procedure BFS_TRAVERSAL ($\Delta$, $S^0$)
  $from^0 = reached = S^0$
  for ($i = 1; ; i++$) {
    $to^i = \text{IMAGE}(\Delta, from^{i-1})$
    $new^i = to^i \cdot \overline{reached}$
    if ($new^i = 0$) then
      return ($reached$)
    $reached = reached + new^i$
    $from^i = new^i$
  }
```

Fig. 1. *Symbolic FSM traversal algorithm*

3.2 FSM markovian analysis

The Markovian behavior of a finite state machine can be studied by regarding its transition structure as a Markov chain. It is sufficient to label each out-going edge of each state with the probability for the FSM to make that particular transition to obtain a discrete-parameter Markov chain. On the other hand, studying the behavior of the Markov chain, that is, computing the state occupation probabilities, is related to performing the reachability analysis of a FSM.

Given the next state (Δ) and output (Λ) functions of the FSM representing the sequential circuit, it is possible to compute the value of the steady-state occupation probability of each state $s \in S$, i.e., the probability that the FSM, when in its steady-state, is in state $s \in S$. For mid-sized circuits the calculation can be carried out in an exact fashion using the ADD-based procedures of [13]; for large sequential networks, the approximate techniques of [14] must be employed. In both cases, complex primary input probability distributions can be specified in order to have more detailed hardware modeling options.

4 Application to protocol verification

Computer-aided design tools for symbolic manipulation and Markovian analysis of finite state machines usually assume the availability of the structural description of a sequential circuit whose functional behavior perfectly matches that of the FSM under analysis. (Notice that the correspondence between circuit implementation and FSM is not unique, that is, a given FSM can be realized by different circuits.) To exploit the capabilities of such tools in the context of protocol verification, a preliminary step of quick (or *low-effort*) synthesis is required to generate a gate-level implementation (called in the following the *protocol equivalent circuit*) of the FSM describing the protocol.

In this section we first consider the case of isolated protocol layers. We illustrate how Markovian verification can be performed on these kind of entities, and we show how the technique can be used in practice in the case of a simple example, namely, the Alternating Bit Protocol. Then, we outline extensions of the method to multi-layer protocol stacks and complete communications systems.

4.1 Analysis and verification of a single protocol entity

Let us assume that a gate-level description of the equivalent circuit for the protocol entity we intend to verify is available. Such description is usually obtained through common high-level and logic-level synthesis tools. The primary input and primary output signals of the circuit model the interface of the protocol entity under analysis with either the neighboring protocol layers, as shown in Fig. 2, the user, or the physical communication channel.

Starting from the gate-level description of the circuit, we initially extract the corresponding state transition graph (STG), and then we identify its topology by STG structural analysis. To perform such analysis, we first calculate the set of reachable states using the symbolic traversal algorithms described in Section 3.1. Typical values of the input transition probabilities are then determined by functional *simulation* of a system that contains the protocol entity under verification and that realistically models the environment in which the protocol entity will run in practice. Alternatively, the designer sets those probability values

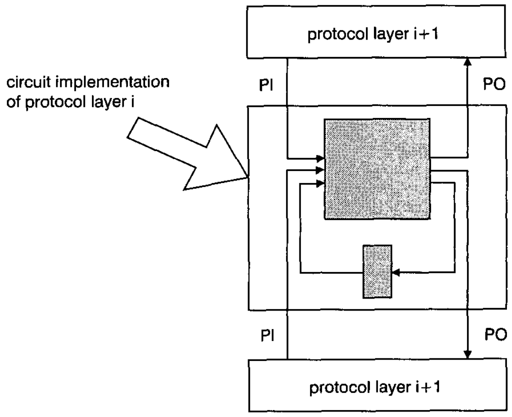

Fig. 2 *Signal interface of the circuit modeling a protocol layer*

according to his/her knowledge about the communication system. Such probabilities are finally propagated through the STG to obtain the state-transition probabilities, which are stored as labels on the edges of the state transition graph.

The STG has now become a weighted directed graph; consequently, as discussed in Section 3.2, it can be regarded as a discrete-parameter Markov chain, on which the terminal strongly connected components are determined by applying the procedure presented by Matsunaga *et al.* in [15]; such procedure calculates the transitive closure of a transition relation. Finally, the steady-state occupation probabilities for the Markov chain are computed using the symbolic calculation methods briefly summarized in Section 3.2.

At this point, the property we are interested in analyzing on the protocol entity is specified in terms of some of the reachable states of the Markov chain on which it should hold, and the probability for a protocol entity to be in such states is computed through symbolic operations. In particular, if (s) represents the set of states identifying the property to be checked and $p(s)$ indicates the steady-state occupation probability of the states in such set, the probability that the property holds is computed symbolically through arithmetic existential abstraction as:

$$Prob(\upsilon) = \backslash_s^+(p(s) \cdot \upsilon(s))$$

This result provides us with an answer about the probability that the protocol entity under verification satisfies the desired property.

4.2 An example: the alternating bit protocol

To better understand how Markovian analysis and verification works, we apply it to a simple example, namely, the alternating bit protocol (ABP) [16], which is a reliable layer-2 (data-link) protocol, according to the ISO/OSI reference model [17].

The protocol works as follows. A sender entity communicates with a corresponding receiver entity exploiting the services of the layer-1 entity, as shown in Fig. 3. The sender and the receiver reliably exchange data in the form of bits having alternatively the values 0 and 1. The receiver acknowledges the reception of each bit; the acknowledgment specifies the value of the received bit. The sender waits for the acknowledgment of the last sent bit; if, when an acknowledgment is received, the specified value differs from the expected one, the last bit is retransmitted.

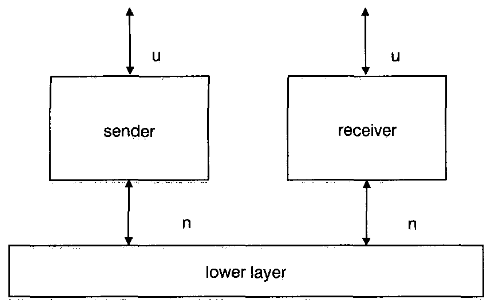

Fig. 3 *ABP: system architecture*

In the following, we consider the sender entity of the ABP protocol. Using the ISO/OSI terminology, the interfaces through which the layers interact are called service access points (SAPs); in our case, they are given the names of *u* (SAP towards the user) and *n* (SAP towards the underlying layer). The service primitives through the *u* interface are *u.Send.Request* initiated by the user and *u.Send.Confirm* originated by the sender entity upon reception of the acknowledgment related to the bit sent to fulfill the last request.

For the sake of simplicity, we assume that the layer-1 protocol entity provides services for sending bits and two kinds of acknowledgments, i.e., it builds, transfers, and interprets the protocol data units used to carry data bits and acknowledgments, thus relieving the ABP from this task. Then, the service primitives through the *n* interface are *n.DT_0.Request* and *n.DT_1.Request* initiated by the sender entity, and *n.Ack_0.Indication* and *n.Ack_1.Indication* initiated by the receiver entity. *n.Ack_0.Indication* means that the receiver has received a bit with value 0 and it has issued a *n.Ack_0.Request* that has caused an acknowledgment message to be sent on the network.

The complete behavior of the ABP sender entity is best summarized by the finite state machine of Fig. 4.

The sender is initially in its reset state, *Ready_0*. In this state, the only event to which the protocol reacts is *u.Send.Request* from the user: The FSM goes to state *WFAck_0* and it issues a *n.DT_0.Request* as related action. In state *WFAck_0*, *n.Ack_0.Indication* tells that the last bit (that had value 0) has been successfully received. Thus, the sender notifies the user by issuing a *u.Send.Confirm* and moves to the state *Ready_1*. When the sender is in this state, the next *u.Send.Request* causes the sending on the network of a bit with value 1, and the FSM goes to state *WFAck_1* issuing a *n.DT_1*.Request. From here, the *Ready_0* state is reached again upon reception of a *n.Ack_1.Indication* meaning that the last bit (that had value 1) has been successfully received.

Given the FSM specification of the sender entity, one property we may be interested in checking is, for instance, the probability of the sender to be in a state waiting for a *n.Ack_0.Indication*. More specifically, we would like to check how such probability evolves as the protocol's boundary conditions change.

We study the aforementioned property by applying Markovian verification. Therefore, we first construct the ABP sender equivalent circuit by synthesising its FSM specification. The resulting circuit, obtained through the SIS [18] logic synthesis tool, has the interface shown in Fig. 5 (each I/O signal represents one service primitive); it consists of a total of 19 logic gates (the circuit has been mapped onto a library containing NAND, NOR, and inverter gates, each of which allows up to four inputs), and its schematic is depicted in Fig. 6.

At this point, we can run the Markovian analysis tool on the ABP sender equivalent circuit. We control the protocol's boundary conditions by setting different transition probabilities at the inputs of the circuit. In particular, we tune the probability values to be assigned to the *u.Send.Request* primitive from the user and to the *n.Ack_0.Indication* primitive from the underlying layer.

The results of our analysis are shown in the diagram of Fig. 7, where on the *x* axis we report the probability of the *u.Send.Request* signal to be active, and on the *y* axis we report the probability of the protocol to be in state *WFAck_0*. Each curve in the diagram is drawn for a different value of the probability of *n.Ack_0.Indication* to be issued by the lower protocol layer.

The diagram confirms that, for a fixed value of the probability of *u.Send.Request*, the probability of state *WFAck_0* decreases for increasing values of the probability of *n.Ack_0.Indication*. When the latter is equal to 1 (the corresponding curve coincides with the *x* axis), we have the extreme situation of the protocol leaving immediately state *WFAck_0*. On the other hand, when the probability of *n.Ack_0.Indication* is equal to 0, no acknowledgment is received, and the protocol is stuck in state *WFAck_0*. Similarly, for a fixed value of the probability of *n.Ack_0.In-*

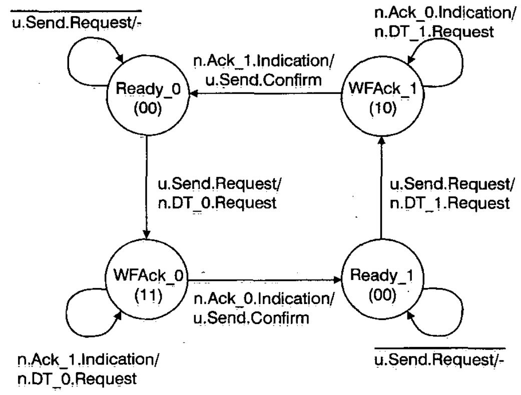

Fig. 4 *ABP: finite state machine of the sender entity*

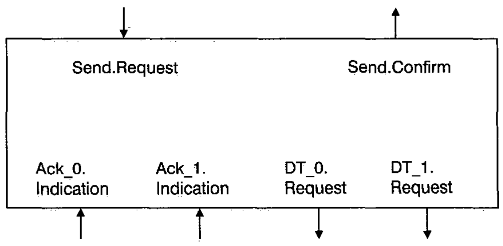

Fig. 5 *ABP: interface of the sender equivalent circuit*

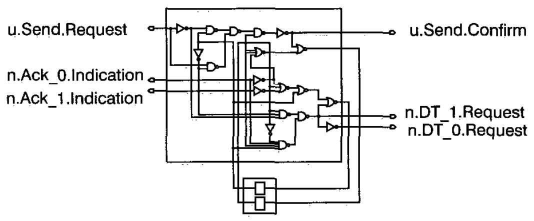

Fig. 6 *ABP: sender equivalent circuit*

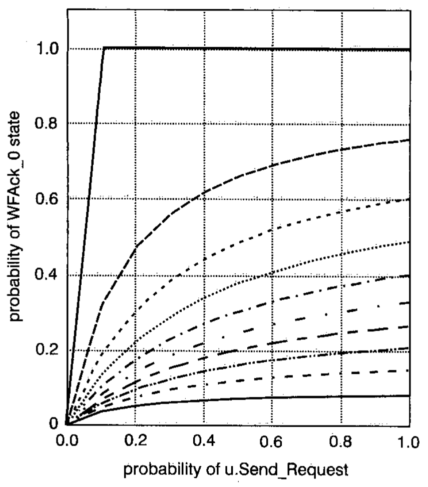

Fig. 7 *ABP: probabilistic analysis of the sender entity*

"n.Ack_*0*.Indication = 0.0" ——
"n.Ack_*0*.Indication = 0.1" – – –
"n.Ack_*0*.Indication = 0.2" - - - -
"n.Ack_*0*.Indication = 0.3" · · · · ·
"n.Ack_*0*.Indication = 0.4" –·–·
"n.Ack_*0*.Indication = 0.5" –·–·
"n.Ack_*0*.Indication = 0.6" - - - -
"n.Ack_*0*.Indication = 0.7" - - - -
"n.Ack_*0*.Indication = 0.8" - - - -
"n.Ack_*0*.Indication = 0.9" ——
"n.Ack_*0*.Indication = 1.0" – – –

dication, the probability of state *WFAck_0* increases for increasing values of the probability of *u.Send.Request*. This is intuitive because, for a given number of acknowledgments received, a larger number of requests implies a longer stay in state *WFAck_0*.

From the diagram it can also be observed the joint effect of the probabilities of *u.Send.Request* and *n.Ack_0.Indication* on the occupation probability of state *WFAck_0*. In particular, for lower values of the probability of *u.Send.Request*, an increased probability of signal *n.Ack_0.Indication* corresponds to a decreased value of the probability of state *WFAck_0*. This indicates that, for low request rates, the effect of the acknowledgment rate on the probability of state *WFAck_0* is dominant. Conversely, for higher request rates, the distance between the curves tends to remain constant, meaning that the effects of the two input conditions on the probability of state *WFAck_0* are much less related.

4.3 Verification of protocol stacks and communication systems

The method of Section 4.1 can be exploited to analyse entire protocol stacks and complete communication systems. In fact, it is sufficient to separately construct the hardware models (i.e., the equivalent circuits) for all the entities in the system, to connect them to build a global equivalent circuit, and to run on it Markovian verification. Obviously, the complexity of the circuit, and therefore that of the STG which is extracted from it, is usually high (e.g., in real cases, the extracted STG may have billions of reachable states). However, thanks to the capabilities of existing Markovian analysis tools (they can easily handle finite state systems with over 10^{30} states and over 75 primary inputs), the method is viable and provides reliable results in reasonably short computation times.

As an application example, let us consider the interaction between the sender and the receiver entities of the ABP. The two equivalent circuits are synthesised; also, a model of the layer-1 protocol entity emulating the low-level services is implemented. The three circuits are then connected through the proper inputs and outputs, and the interaction between sender and receiver is studied by analyzing the resulting global equivalent circuit, shown in Fig. 8. The circuit takes into account errors and losses in the lower layer. When performing Markovian analysis, a non-null probability on the input named *Error* allows the behavior of the protocol to be studied in presence of errors in the services provided by the lower layer. The analysis of the circuit with a non-null probability assigned to the input named *Loss* yields a probability of 1 for the sender entity to be in state *WFAck_0* or *WFAck_1*. This high-lights that the protocol cannot be used in lossy environments due to the lack of time-out mechanisms.

5 Conclusions

We have presented an approach to verification of communication protocols based on Markovian FSM analysis.

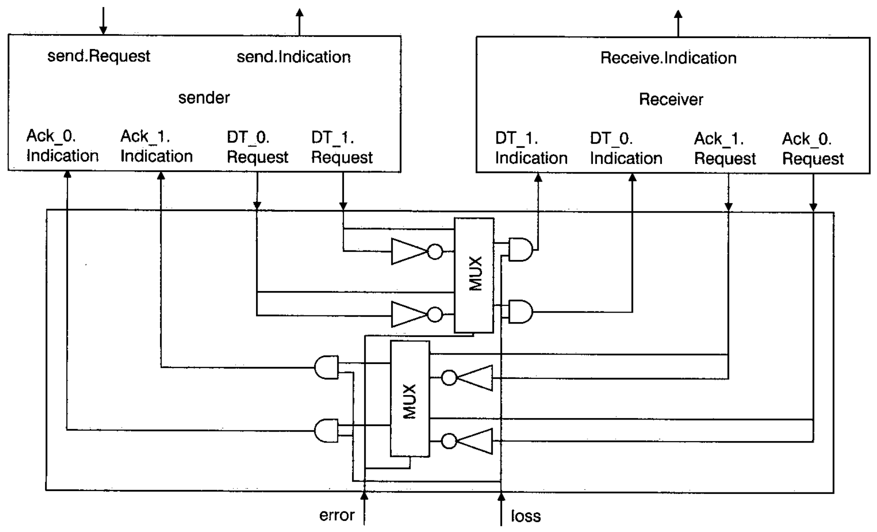

Fig. 8 *ABP: interaction between sender and receiver*

Thanks to the availability of efficient tools developed in the context of logic synthesis the presented method is applicable to real communications systems, whose finite state models may contain billions of states.

We have first described the technique in the case where a single protocol entity is to be considered. Then, we have shown how the method works on a simple example, namely, the sender entity of the alternating bit protocol. Finally, we have illustrated how the approach can be extended to the case of more complex protocols, to entire protocol stacks and to complete communication systems.

6 References

1 KEMENY, J., and SNELL, J.: 'Finite Markov chains' (D. Van Nostrand Company 1967)

2 TRIVEDI, K.S.: 'Probability and statistics with reliability, queueing, and computer science applications' (Prentice-Hall, 1982)

3 BRYANT, R.: 'Graph-based algorithms for boolean function manipulation,' *IEEE Trans. Comput.*, 1986, **C-35**, (8), pp. 79–85

4 CLARKE, E.M., FUJITA, M., MCGEER, P.C., MCMILLAN, K.L., and YANG, J.: 'Multi-terminal binary decision diagrams: an efficient data structure for matrix representation', IEEE International Workshop on *Logic Synthesis*, IWLS-93, May 1993, Lake Tahoe, CA, pp. 6a:1–15

5 BAHAR, R.I., FROHM, E., GAONA, C., HACHTEL, G.D., MACII, E., PARDO, A., and SOMENZI, F.: 'Algebraic decision diagrams and their applications', *Formal Methods in System Design*, 1997, **10**, pp. 171–206

6 BAIER, C., CLARKE, E.M., HARTONAS-GARMHAUSEN, V., KWIATKOWSKA, M., and RYAN, M.: 'Symbolic model checking for probabilistic processes', Automata, Languages and Programming, 24th International Colloquium, ICALP-97 Proceedings, 1997 (Springer-Verlag, Berlin, Germany, pp. 430–440)

7 HUTH, M., and KWIATKOWSKA, M.: 'Quantitative analysis and model checking', IEEE Annual Symposium on *Logic in Computer Science*, 1997, pp. p.111–122

8 COUDERT, O., BERTHET, C., and MADRE, J.C.: 'Verification of sequential machines using boolean functional vectors', IFIP Intl. Workshop on *Applied Formal Methods for Correct VLSI Design*, November 1989, Leuven, Belgium, pp. 111–128

9 COUDERT, O., and MADRE, J.C.: 'A unified framework for the formal verification of sequential circuits', IEEE International Conference on *Computer-Aided Design*, ICCAD-90, November 1990, Santa Clara, CA, pp. 126–129

10 TOUATI, H., SAVOJ, H., LIN, B., BRAYTON, R.K., and SANGIOVANNI-VINCENTELLI, A.: 'Implicit enumeration of finite state machines using BDD's', IEEE International Conference on *Computer-Aided Design*, ICCAD-90, November 1990, Santa Clara, CA, pp. 130–133

11 CHO, H., HACHTEL, G.D., JEONG, S.W., PLESSIER, B., SChWARZ, E., and SOMENZI, F.: 'ATPG aspects of FSM verification', IEEE International Conference on *Computer-Aided Design*, ICCAD-90, November 1990, Santa Clara, CA, pp. 134–137

12 BURCH, J.R., CLARKE, E.M., MCMILLAN, K.L., and DILL, D.L.: 'Sequential circuit verification using symbolic model checking', ACM/IEEE Design Automation Conference, DAC-27, June 1990, Orlando, FL, pp. 46–51

13 HACHTEL, G.D., MACII, E., PARDO, A., and SOMENZI, F.: 'Markovian analysis of large finite state machines', *IEEE Trans Comput.-Aid. Des. Integr. Circuits Syst.*, 1996, **CAD-15**, (12), pp. 1479–1493

14 TSUI, C.Y., MONTEIRO, J., PEDRAM, M., DEVADAS, S., DESPAIN, A.M., and LIN, B.: 'Power estimation methods for sequential logic circuits', *IEEE Trans. VLSI Syst.*, 1995, **VLSI-3**, (3), pp. 404–416

15 MATSUNAGA, Y., MCGEER, P.C., and BRAYTON, R.K.: 'On computing the transitive closure of a state transition relation', ACM/IEEE Design Automation Conference, DAC-30, June 1993, Dallas, TX, pp. 260–265

16 SARIKAYA, B.: 'Principles of protocol engineering and conformance testing' (Ellis Horwood, 1993)

17 International Standards Organization, 'IS-7498: Information processing systems, open systems interconnection, basic reference model', 1984, Geneva, Switzerland

18 SENTOVICH, E.M., SINGH, K.J., MOON, C.W., SAVOJ H., BRAYTON, R.K., and SANGIOVANNI-VINCENTELLI, A.: 'Sequential circuits design using synthesis and optimization', IEEE International Conference on *Computer Design*, ICCD-92, October 1992, Cambridge, MA, pp. 328–333