MODELLING AND ANALYZING DYNAMIC SOURCE ROUTING PROTOCOL WITH GENERAL DISTRIBUTIONS

Osman Salem
Abdelmalek Benzekri
Institut de Recherche en Informatique de Toulouse
Université Paul Sabatier
118 Route de Narbonne, 31062 Toulouse Cedex 04, France
E-mail: {benzekri, osman}@irit.fr

KEYWORDS
SPA, CTMC, Performance Evaluation, General distributions, Insensitivity, MANETs, DSR.

ABSTRACT

In this paper, we present an integrated algebraic model of the dynamic source routing protocol used in ad hoc networks. Algebraic description of this protocol has been realized by means of powerful operators of Extended Markovian Process Algebra EMPA and by exploiting value passing feature in order to express activities execution delay with general distribution. Afterwards, functional properties of the model are formally verified using μ-calculus/CTL model checking formulas and performance parameters are assessed via simulation through the EMPA software tool.

1. INTRODUCTION

Modelling and analyzing a concurrent system, especially when it is composed of a large number of components that cooperate to achieve some tasks, are very important to prevent costly redesigns and time loss (Bernardo et al. 1998) by verifying functional and performance properties of the model early in design phase and before its implementation (Bernardo and Gorrieri 1996).

Classical process algebras like CCS (Milner 1989) and CSP (HOARE 1985) are formal description techniques that provide a linguistic means: to model a system by describing it mathematically via predefined operators and to analyze its characteristics in order to ensure that it is correctly designed (Brinksma and Hermanns 2001). This modelling formalism allows designers to specify only the functional aspects of the system and to verify its qualitative properties by analyzing the labeled transition graph derived from its algebraic description.

However, neglecting the temporal aspect of the system's behaviour is a major drawback in the expressiveness of this formalism, because once the model is realized or implemented and it seems that it does not satisfy the required performance criteria, it must be redesigned (Bernardo et al. 1998; Benzekri 2002). To alleviate this problem, the designer models quantitative properties by resorting to stochastic processes such as Markov chain. Using this technique, the designer describes what possible states the system may enter and how it moves form one state to another in time. However, this specification becomes tedious, error prone and manually infeasible when the state space of the model contains many hundreds of states, as it is often the case in practice. Furthermore, tests for functionality and performance are realized on two different models of the system, so the designer has to make sure that these models are consistent.

In response to these limitations, Stochastic Process Algebras (SPA) (Brinksma and Hermanns 2001; Benzekri 2002) have been introduced as an extension to classical process algebras via the assignment of an exponentially distributed random variable to each activity in the model representing its duration. Time is restricted to exponentially distributed because the underlying labeled transition graph can easily be transformed into a Continuous Time Markov Chain (CTMC for short), which is limited to describe systems satisfying the property that the future behaviour only depends on the current state.

SPAs provide then an integrated formalism where functional and performance aspects are both taken into account from the beginning of the design process, thereby achieving a part of the desired objective. Beside this, it inherits compositionality and abstraction features of classical process algebras. These attractive features allow user to construct a model from smaller components without considering their internal structure. In the last years, several SPA languages and tools have appeared, such as PEPA (Performance Evaluation Process Algebra (Hillston and Ribaudo 1996)), TIPP (TImed Processes and Performability evaluation (Götz et al. 1993)) and EMPA (Extended Markovian Process Algebra (Bernardo 1998)). All these languages propose the same approach to performance modelling by restricting time to exponential for deriving a CTMC from the described algebraic model, and then they use linear algebra methods to get steady states and transient states distribution probabilities, which are useful to assess performance parameters. However, the most significant difference between these stochastic process algebras is related to synchronization rate of processes interaction (proofs are detailed in (Brinksma and Hermanns 2001)), which results when two or many components synchronise, or cooperate, to achieve an action.

Among these languages, we will focus in this paper at EMPA, which was inspired from existing languages (PEPA and TIPP). It extends the expressiveness of these languages by allowing the designer to describe prioritized weighted immediate actions with exponentially distributed actions. The restriction in describing activities duration to exponential was necessary in SPA in order to define the semantics of these languages in an interleaving style like in classical process algebras by exploiting the memory-less property, which makes the derivation of CTMC easily from label transition diagram by discarding the activities name and by keeping only the temporal parameters (Hillston and Ribaudo 1996; Bernardo 1998).

The restriction of activity timing aspects to exponential distribution is regarded by some like a limitation in the expressiveness power of SPA modelling formalism, because even deterministic delay, which frequently arises in practice, can not be specified (e.g. time out in communication protocols).

The combined use of immediate and exponentially delayed actions in EMPA, allows phase type distribution (Bernardo 1996) to be represented, which is very useful because many frequently occurring distributions are or can be approximated by phase type distributions. Well-known examples of phase type distributions are: hypo-exponential, hyper-exponential and coxian distribution.

Indeed, several attempts have been done to incorporate general distributions in this modelling formalism, to make it able to express directly durations with arbitrary distributions and to avoid approximation through combination of actions. However, general distributions approach lead to intractable model where it is often impossible to analyse it. Also, The memory-less property can not be exploited to define the semantics in interleaving style because the actions can no more be thought as being started in the states where they will be executed. In contrast, we must keep track of the sequence of states in which an action started and continued its execution. The underlying performance diagram is no longer Markovian and its performance can be evaluated by using mathematical methodologies in some cases by employing the notion of insensitivity (Clark 1999), or by using simulation.

A stochastic process is said to be insensitive if its steady state distribution depends on the distribution of one or more of its state lifetime random variables only through their mean. This means that these random variables may be replaced with other distributed ones having identical mean for preserving the steady state solution of the process. As a consequence exponential distributions can be used to transform the underlying transition graph into CTMC that can be easily analysed.

Bravetti et al. in (Bravetti et al. 1997), propose a Generalised Semi Markov Process Algebra GSMPA modelling formalism, which incorporates general probability distributions. This modelling formalism includes all operators used in process algebra. To keep track of residual lifetimes of actions in execution through many states, they adopt a mechanism of action identification by representing each action in the semantic model as a combination of action start and action termination. In addition, they rely at preselection policy to resolve choice according to priority level and weight associated to each activity. Afterward, they provide a mapping to GSMP and exploit insensitivity to resolve it. In (Bravetti et al. 1998), they give an example of a simple queue modelled in GSMPA, where the service time was deterministic, and then they exploit insensitivity to derive CTMC from GSMP by aggregating particular states.

Bernardo in (Bernardo 1997) enhances the expressiveness power of EMPA by adding the mechanism of value passing to this formalism. In this paper, we will model the DSR protocol with general distributions by exploiting value passing strategy, where this protocol will be expressed in the algebraic formal description language EMPA_{vp} to describe general distributed activities.

The rest of this paper is organized as follows. In section 2 we briefly introduce the syntax of EMPA with value passing and the meaning of its operators. In Section 3 we present the basic operations of DSR with the algebraic description of its items. In section 4 and 5, we present the results of the functional and performance analysis realized at the DSR algebraic model. In section 6 we report some concluding remarks.

2. EXTENDED MARKOVIAN PROCESS ALGEBRA: EMPA

In this section, we present the syntax of the specification language EMPA that is necessary to understand the algebraic description of the DSR protocol and we concentrate on features such as value passing, general distributions and simulation that we have exploited in our case study.

EMPA allows the description of a complex system from many components modelled separately and then combined using the appropriate operators. Every component performs timed activities represented by $<a, \lambda>$, where "a" characterizes action name, and "λ" represents action duration given by the rate of exponential distribution function $F(t) = 1-e^{-\lambda t}$ and indicates the speed at which the action occurs. Based on the value of its rate, an action is classified as exponentially timed if "λ" is a positive real number, or passive if "λ" is left unspecified (denoted *) usually used to model activities waiting for synchronization, or immediate if "λ" is equal to infinity. Immediate actions represented by $<a, \infty_{L,W}>$, have two parameters: L and W. the former is used to express the priority level of action "a" and the second is used to express the weight of action execution. EMPA processes can be constructed according to the following syntax:

$$P = 0 \mid <a,\lambda>.P \mid <a,*>.P \mid <a ,\infty_{L,W}>.P \mid P/L \mid P[\varphi] \mid P + Q \mid P \parallel_s Q \mid A$$

$$VP = <a!(x), \infty_{L,W}>.P \mid <a?(x), \infty_{L,W}>.P \mid \text{if } (\beta,p) \text{ the } P_1 \text{ else } P_2 \mid A(\text{local\_param} ; \text{local\_var})$$

Here 0 denotes the zero process that cannot perform any actions. Expression <a,->.P represents sequential execution of action "a" followed by process P. The hiding operator -/L transforms actions belonging in list L into internal actions whose functionality cannot be observed (like τ in LOTOS (Bolognesi and Brinksma 1987)). The re-labeling operator $P[\varphi]$ renames actions in process P according to φ equation. The choice operator P + Q executes process P or process Q depending on whether an action in P or in Q is executed first. The parallel composition operator $P \parallel_S Q$ allows the asynchronous execution of P and Q activities, which are not belonging to list S and synchronous execution at actions listed in S, where synchronization can take place only between active and passive actions. Finally, constant operator is used to express recursive behaviour.

Implementing value passing in EMPA, add to unstructured actions of the form $<a, \lambda>$ additional information needed to exchange data among system components which synchronize when performing some activities. Actions which take data variable x as input are of the forms $<a?(x), \lambda>$ and actions that output data expression e are of the forms $< a!(e), \lambda>$. Further details can be found in (Aldini et al. 2001).

The conditional operator (if (β, p) then P_1 else P_2) has been added to EMPA, where β is a Boolean expression and p is the probability that β is satisfied. In simulation, the boolean expression is considered to solve the choice, while the probability p that β holds is used in case of numerical analysis and this operators can be represented by the internal immediate choice like appears in the following expression: $<\tau, \infty_{1, p}>. P_1 + <\tau, \infty_{1,1-p} >.P_2$.

Finally, when using value passing, we should be able to keep track of the data we are interested in, this can be achieved by means of parameterized constant definitions of the form A(x) = P, where x is a vector of variables composed of local variables and formal parameters (Bernardo 1997). Local variables are used to get values from other processes via synchronisation while formal parameters are bound to actual parameters used when a constant invocation occurs.

3. FORMAL DESCRIPTION OF DSR

The dynamic source routing protocol (Broch et al. 2003) is an efficient reactive routing protocol designed for use in Mobile Ad Hoc Networks (MANETs), which are a collection of mobile host dynamically forming a temporary network, without the need for any existing network infrastructure or administration. Due to the limited transmission range of wireless network interfaces, multiple hops may be needed for one node to exchange data with another across the network. Therefore, each mobile node inside such network may operate not only as a host but also as a router, forwarding packets for other mobile nodes in the network that may not be within direct wireless transmission range of each other.

DSR uses source routing rather than hop-by-hop routing (e.g. packets carry in their header the path through which it must pass). The key advantage of source routing is that intermediate nodes do not need to maintain up-to-date routing information in order to route the packets they forward, since packets themselves contain all the routing decisions. This fact, couplet with the on demand nature of the protocol, eliminates the need for the periodic exchange of routing packets.

This protocol allows the network to be completely self-organizing and self-configuring in the following manner: when source host attempts to send packets to another host and does not already know a route to this last, it floods a Route Request packet (RREQ) in the network. Therefore, when an adjacent node receives this packet, it appends its address, and floods the received packet (RREQ) if it is not the destination and does not see this request before. When RREQ packet reaches destination host, it contains in its header the path towards the source, so destination host will use this path to send a route reply packet (RREP) if the link is full duplex. When source host receives RREP, it appends received path to every packet that it will send. Intermediate host will route these packets according to header-included path. If the network topology has changed such that the used path is broken because one host listed in path has moved out, the adjacent host of this last notifies the source by sending Route Error packet (RERR). When source host receives route error packet, it restarts the same algorithm to find another path if it does not have other previously learned route.

The dynamic source routing protocol described in previous section has been formally modeled with EMPA for analyzing its qualitative and quantitative parameters. Using value passing has been necessary to model activities with general distributions (such as deterministic or generally distributed time out). We present the algebraic specification that has been done while exploiting compositionality to deal with three entities: Source, Receiver and MANET. The complete specification of DSR is given by:

```
• DSR 
      $S_0$(nb, req_id, to, ceil(exp(1))) $||_{SM}$ MANET $||_{RM}$ R

• MANET 
      NET_std(clk_std,[],mem) || NET_dts(clk_dts, [])
```

The interaction between the source and the network is given by the list of actions in SM while the interaction between the network and the receiver is given by the list MR.

SM = {rreq, net_rreq, net_rrep, net_error, send_pkt, resend_pkt, net_pkt }
MR = {net_rreq, deliver_rrep, net_pkt}

3.1 Specification of the source

The source host begins by checking its cache to find if there is any previously learned path toward the intended destination, so if it finds a cached path, it appends it to every packet and sends it to next hop. However, if it doesn't find a cached path, it floods RREQ and wait until the reception of RREP. Aware that many RREQ packets flooded by the network may be returned to the source, this last exploits these packets to discover its neighbors. Also after sending a RREQ, if time out occurs before receiving any route reply packet (usually after sending RREQ packet 7 times), the source wait for an amount off time, usually called exponentially back-off, to limit the rate of transmitted RREQ. The specification of this component is the following:

```
• $S_0$(int nb, int req_id, int to, int back_off;) 
    <prep_pkt,  $\infty_{1,1}$>.<verif_cache, $\infty_{1,1}$>.
   (<$\tau$, $\infty_{1,50}$>.<pnf, $\infty_{1,1}$>.$S_1$(nb, req_id, $to_0$, back_off)
+   <$\tau$,$\infty_{1,50}$>.<pf, $\infty_{1,1}$>.Sender(nb, req_id, to, back_off))

$S_1$(int nb, int req_id, int to, int back_off; int rq_id) 
    <rreq!(req_id), $\infty_{1,1}$>.$S_1$'(nb, req_id, to -- 1, back_off)

$S_1$'(int nb, int req_id, int to, int back_off; int n) 
  if (to > 0, 0.8) then
      <net_rreq?(n), *>.<discard_rreq, $\infty_{1,1}$>.
      $S_2$(nb, req_id, $to_0$, back_off)
  else
      <elapse_tick, $\infty_{1,1}$>.
      Check_nb(nb, req_id, to --1, back_off)

Check_nb(int nb, int req_id, int to, int back_off;)  
   if (nb <= 3, 0.99) then
       <rreq!(req_id ++ 1), $\infty_{1,1}$>.
       $S_1$'(nb ++ 1, req_id ++ 1, to -- 1, back_off)
   else
       <back_off_phase, $\infty_{1,1}$>.
       Back_off(1, req_id ++ 1, $to_0$, back_off --1)
Back_off(int nb, int req_id, int to, int back_off;) 
   if (back_off < 0, 0.5) then
       <try_again, $\infty_{1,1}$>.$S_0$(1, req_id, $to_0$, ceil(exp(1)))
   else
       <elapse_tick, $\infty_{1,1}$>.
       Back_off(nb, req_id, to, back_off -- 1)

$S_2$(int nb, int req_id, int to, int back_off; int rrepl) 
   if (to >> 0, 0.9) then
       <net_rrep?(rrepl),*>.
       Verif_rrep(rrepl, nb, req_id, to -- 1, back_off)
     + <elapse_tick,$\infty_{1,1}$>.$S_2$(num,req_id,to -- 1, back_off)
   else
       <timeout, $\infty_{1,1}$>.$S_1$(nb, req_id ++ 1, $to_0$, back_off)

Verif_rrep(int rrepl, int nb, int req_id, int to, int
back_off;) 
   if (req_id >= rrepl, 0.8) then
       <cache_path, $\infty_{1,1}$>.<add_path2pkt, $\infty_{1,1}$>.
       Sender(1, req_id, $to_0$, back_off)
   else
       <discard_rrep, $\infty_{1,1}$>.$S_2$(nb, req_id, to, back_off)

Sender(int nb, int req_id, int to, int back_off;) 
       <send_pkt, $\infty_{1,1}$>.
       Hear_forward(1, req_id, $to_0$, back_off)

Hear_forward (int nb, int req_id, int to, int back_off; )
   if (to > 0, 0.9) then
       (<net_pkt, *>.Sender(nb, req_id, $to_0$, back_off)
     + <elapse_tick, $\infty_{1,1}$>.
       Hear_forward(nb, req_id, to -- 1,back_off))
   else
       <elapse_tick, $\infty_{1,1}$>.
       Check_sended(nb, req_id, to -- 1, back_off)

Check_sended(int nb, int req_id, int to, int back_off;) 
   if (nb <= 3, 0.5) then
       (<resend_pkt, $\infty_{1,1}$ >.
        Hear_forward(nb ++1, req_id, $to_0$, back_off)
     + <net_error, *>.
        Check_sended(nb, req_id, to -- 1, back_off)
     + <net_pkt,*>.Sender(nb, req_id, $to_0$, back_off))
   else
       <broken_net, $\infty_{1,1}$>.$S_0$(1, REQ_ID, $to_0$, back_off)
```

3.2 Specification of the network

The MANET-std and MANET-dts components model the mobile network from source to destination and from destination to source respectively, which is composed from many intermediate nodes. This entity may receive RREQ from the source host or directly packets that must forwarded to the receiver host. Also it may: floods received route request packet, sends received route reply packet to source host, move outside source transmission zone, deliver route request to receiver host and sends route error packet to the source when receiver become unreachable. The complete specification of DSR is given by:

```
• MANET_std(int clk_std, list(list(int)) list_pkt, int
mem; int id) 
   <RREQ?(id),*>.
           Verify_ID(clk_std ++ 1, list_pkt, mem, ID)
+ <send_pkt, *>.
           Delivery_check_pkt(clk_std ++ 1,
           insert([clk_std ++ ceil(normal(100,7))],
           list_pkt), mem)
+  <resend_pkt, *>.
           Delivery_check_pkt(clk_std ++ 1,
           insert([clk_std ++ ceil(normal(100,7))],
           list_pkt), mem)
+  <$\tau$, $\infty_{1,0.01}$>.<move_out, $\infty_{1,1}$>.
           MANET_std(clk_std ++ 1, list_pkt, mem)

Delivery_check_pkt(int clk_std, list(list(int)) list_pkt,
int mem;) 
if ((list_pkt != []) &&
  (clk_std >= first(first(list_pkt))), 0.9) then
        (<$\tau$, $\infty_{1,0.99}$>.<net_pkt, $\infty_{1,1}$>.
        MANET_std(clk_std ++1, tail(list_pkt), mem)
        + <$\tau$, $\infty_{1,0.01}$>.<net_error, $\infty_{1,1}$>.
        MANET_std(clk_std ++1, [], mem))
else
        <elapse_tick, $\infty_{1,1}$>.
        Delivery_check_pkt(clk_std ++ 1, list_pkt, mem)

Verify_ID(int clk_std, list(list(int)) list_pkt, int mem, int
ID;) 
if (id >= mem, 0.9 ) then
        (<add_cache, $\infty_{1,1}$>.Delivery_rreq(clk_std ++ 1,
        insert([clk_std ++ ceil(normal(100,7))], list_pkt),
        ID)
        + <$\tau$, $\infty_{1,0.001}$>.discard_rreq, $\infty_{1,1}$>.
        MANET_std(clk_std ++ 1, list_pkt, mem))
else
        <discard_rreq, $\infty_{1,1}$>.
        MANET_std(clk_std ++ 1, list_pkt, mem)

Delivery_rreq(int clk_std, list(list(int)) list_pkt, int
mem;) 
if (clk_std >= first(first(list_pkt)),0.9) then
        <net_rreq!(mem), $\infty_{1,1}$>.
        MANET_std(clk_std ++ 1, tail(list_pkt), mem)
else
        <elapse_tick,$\infty_{1,1}$>.
        Delivery_rreq(clk_std ++ 1, list_pkt, mem)

• MANET_dts(int clk_dts, list(list(int)) delivery_time;
int rep_id) 
      <deliver_rrep?(rep_id),*>.
      Delivery_rrep(clock_dts ++ 1, insert([clk_dts ++
      10], delivery_time), rep_id)

Delivery_rrep(int clk_dts, list(list(int)) delivery_time,
int rep_id;) 
if (clock_dts >= first(first(delivery_time)), 0.9) then
      <NET_RREP!(rep_id), $\infty_{1,1}$>.
      MANET_dts(clk_dts ++ 1, tail(delivery_time))
  else
      <elapse_tick, $\infty_{1,1}$>.
      MANET_dts(clock_dts ++ 1, delivery_time)
```

3.3 Specification of the receiver

The receiver host may receive many route request packets and it answers to all received packets. The source uses only the first received one by supposing that it crosses the best route and experiences the less delay, and it drops all others. Afterwards, it delivers received packets to the appropriate applications. The specification of this component is given by:

```
• R(; int r_id) 
     <net_rreq?(r_id),*>.R$_1$(r_id)
  + <net_pkt, *>.<del_to_appl, $\infty_{1,1}$>.R

R$_1$(int r_id$_1$; int new_id) 
     <deliver_rrep!(r_id$_1$), $\infty_{1,1}$>.R
+    <net_rreq?(new_id),*>.R$_1$(new_id)
+    <net_pkt, *>.<del_to_appl, $\infty_{1,1}$>.R
```

4. FUNCTIONAL ANALYSIS

The functional analysis aims at verifying the correctness of the designed model and at detecting conceptual errors in its behaviour. We start our analysis by verifying freedom from deadlock, e.g. states without outgoing transitions also called absorbing states. A naïve strategy would be to use the simulation approach and consider several simulation runs. As the coverage of such simulations is some time low, e.g. if no deadlock is reached during simulation, this does not guaranteed absence of deadlocks.

In EMPA, freedom from deadlock can be verified easily by generating the state space of the algebraic specification of DSR, which has 1825 states (0 tangible, 1825 vanishing, 0 open, 0 deadlocked) and 3888 transitions. Note that the state spaces are very compact because the semantic models are symbolic. Due to space constraints, we cannot present in detail EMPA and its theory; we invite the reader to see for example (Bernardo 2003) for further details.

We have used μ-calculus/CTL (Clarke et al. 1986) model checking to verify other behavioural aspects of our algebraic model. More precisely, we have proved that: the receiver does not send any RREP packet before the network deliver a RREQ packet, and the network does not flood Route Request packet before the sender sends a Route Request. These properties have been formalized through the following formulas:

- (AG([deliver_rrep]
  A([deliver_rrep]ff W <net_rreq> tt)))

- (AG([net_rrep] A([net_rrep]ff W < rreq> tt)))

The first (second) equations means that for any state (operator G) of any computation (operator A) starting at the initial state, action deliver_rrep (net_rrep) can be executed by a state where its derivative must verify the following condition: a deliver_rrep (net_rrep) labelled transition cannot be encounter before a net_rreq (rreq) labelled transition is executed. They check the order of actions' executions.

5. PERFORMANCE ANALYSIS

The performance analysis, which aims at determining efficiency parameters, has been made by using simulation routine in EMPA software tool (TwoTowers), because the presence of generally distributed activities encoded within assignments, which cannot be taken into account elsewhere and an exact performance analysis realised at Markov chain would be meaningless, because assignments are kept as symbolic (Bernardo 2003).

The simulation routine implemented in EMPA tool is based at the method of independent replications, which means that in each step of each simulation run, the transitions for the current state are computed according to the formal semantic and one of them (together with the related derivative states) is chosen.

To realize this simulation, we use an auxiliary specification containing additional information like the termination condition, the number of simulation runs and the performance parameters of interest. In our study, the termination condition was:

$$\text{elapse\_tick } n_1 \; n_2$$

Where elapse_tick is the name of action representing the passage of 1ms, n_1 is the number of times action elapse_tick must be executed before terminating each simulation run, and n_2 is the number of simulation runs. Performance parameters are calculated by using reward, which is a simple method that associates a value to related performance parameters. The interested reader is referred to (Bernardo 1996) for more details about rewards.

Many performance parameters can be calculated (like throughput, utilization rate …), but we are interested to the ratio of routing packets (overload percentage that DSR adds to network due to energy, CPU and memory limitations of intermediate hosts) and to percentage of delivered packets. In order to get these parameters, we have realized the simulation by specifying n_1 to 100000 ms (100s) and n_2 to 20 simulation runs. The results are reported in table 1, where we can see that in worst case in our model, only 6% of packets are added by DSR in the beginning of transmission and the average value during all the transmission time was 3.12%. These results are directly related to the numerical values of probabilities about the speed of each host (very small in our model) and packets loss rate in the network, which are represented in our model by the probability of generating route error and by an internal action (τ) that represents loss.

Table 1: Ratio of Routing Packets

Experiment	estimate	90% confidence int.
estimate 0	5.92705	[5.09417, 6.75993]
estimate 1	3.04501	[2.32435, 3.76568]
estimate 2	2.31663	[1.79883, 2.83442]
estimate 3	3.31035	[2.58241, 4.0383]
estimate 4	3.21337	[2.53473, 3.89202]
estimate 5	2.57794	[1.94359, 3.2123]
estimate 6	2.59094	[2.05932, 3.12255]
estimate 7	3.1949	[2.39891, 3.99088]
estimate 8	2.26937	[1.77628, 2.76246]
estimate 9	2.8377	[2.33404, 3.34135]

Table 2 shows the results obtained for the packet delivery ratio (PDR) given by:

$$\text{PDR} = \frac{\text{nb of data packets received}}{\text{nb of data packets sent}}$$

Table 2: Packet Delivery Ratio

exp.	estimate	90% confidence int
estimate 0	0.988003	[0.986436, 0.989571]
estimate 1	0.990486	[0.988683, 0.99229]
estimate 2	0.991686	[0.990439, 0.992932]
estimate 3	0.989334	[0.985785, 0.991083]
estimate 4	0.989454	[0.987893, 0.991015]
estimate 5	0.991624	[0.989809, 0.99344]
estimate 6	0.991126	[0.989637, 0.992615]
estimate 7	0.990017	[0.98821, 0.991823]
estimate 8	0.991999	[0.990487, 0.993511]
estimate 9	0.9911	[0.990053, 0.992147]

This means that with 100 packets transmitted by the source, 99 packets delivered by the network to receiver host, and these results was expected because the probability of loosing packets is considered equal to 0.01 in our algebraic model.

6. Conclusions and Further Research

In this paper, we have formally modelled and analyzed by means of the stochastic process algebra EMPA, the Dynamic Source Routing protocol with general distributed activities while exploiting the value passing mechanism and the compositionality feature of process algebras to construct a complex model from small components. This case study shows the adequacy and the expressive power of this formalism to model a complex system and to analyze both its functional and its performance properties early in its design phase.

Accuracy of our results is going to be depending on the details that we have invested in the model. Here, we do not claim that DSR is the best routing protocol because the results of described algebraic model are optimal. In contrast, the protocol is not under some circumstances (highly dynamic nature of mobile hosts). However, this protocol is interesting enough to deserve a closer investigation about possible attacks and how to make it secure in cases where it provides the best performance with respect to other existing routing protocols. The focus of our current and future work is to analyze and compare the influence of securing this protocol at its performance, because security mechanisms in this type of networks consume resources and can delay or even prevent successful exchanges of routing information. Many proposals to secure DSR protocol have their own merits and some of them may appear to be extremely secure, but their real performance when deployed in the network is still unclear. Therefore, we will investigate the effect of adding security tasks that we will propose at this protocol by extending our algebraic model to develop a trade off between network performance and security solution that effectively balances security strength and network performance.

References

Aldini, A.; M. Bernardo; R. Gorrieri; and M. Roccetti. 2001. “Comparing the QoS of Internet Audio Mechanisms via Formal Methods”, *in ACM Transaction on Modeling and Computer Simulation,* No.11 (Jan),1-42.

Benzekri, A. 2002. “Qualitative and Quantitative Evaluation using Process Algebra”, The 17th International Symposium on Computer and Information Sciences, (Orlando, Florida USA, Oct. 26-28), 415-418.

Bernardo, M. and R. Gorrieri. 1996. “Extended Markovian Process Algebra”, *in Proceedings of the 7th International Conference on Concurrency Theory*, (Pisa, Italy), Montanari U. & Sassone V. editors, Lecture Notes in Computer Science 1119, 315-330.

Bernardo, M. 1996. “On the Coexistence of Exponential, Immediate and Passive Actions in EMPA“, in *Proceedings of the 4th Int. Workshop on Process Algebras and Performance Modelling*, (Torino, Italy, July), M. Ribaudo editor, 58-76.

Bernardo, M. 1997. “Enriching EMPA with Value Passing: A Symbolic Approach based on Lookahead”, *in Proceedings of the 5th International Workshop on Process Algebras and Performance Modelling*, (Enschede, Netherlands, Jun), E. Brinksma and A. Nymeyer editors, 35-49.

Bernardo, M. 1998 “A Tutorial on EMPA: A Theory of Concurrent Processes with Non-determinism, Priorities, Probabilities and Time”, *in Theoretical Computer Science*, vol. 202 (July), 1-54.

Bernardo, M.; L. Donatiello and R. Gorrieri. 1998. “A Formal Approach to the Integration of Performance Aspects in the Modeling and Analysis of Concurrent Systems”, *International journal of Information and Computation*, Vol. 144, No.2 (August), 120-154.

Bernardo, M. 2003. “Symbolic Semantic Rules for Producing Compact STGLA from Value Passing Process Descriptions”, *ACM Transactions on Computational Logic*.

Bolognesi, T. and Brinksma, E. 1987. “Introduction to the ISO Specification Language LOTOS”, *Computer Networks and ISDN Systems, Elsevier Science Publishers*, Vol. 14, No.1, 25-59.

Bravetti, M.; Bernardo, M. and Gorrieri R. 1997. “Generalized Semi Markovian Process Algebra”, Technical Report UBLCS-97-9, University of Bologna (Italy, Oct).

Bravetti, M.; M. Bernardo and R. Gorrieri. 1998. “Towards performance evaluation with general distributions in process algebras”, *Lecture Notes in Computer Science*, No.1466, 405-422.

Brinksma, Ed. and H. Hermanns. 2001. “Process Algebra and Markov Chains”, *Lectures on Formal Methods and Performance Analysis*, (Nijmegen), 183 - 231.

Broch, J.; D.B. Johnson; and D.A. Maltz. 2003. “The Dynamic Source Routing Protocol For Mobile Ad Hoc Networks”, *IETF draft*, (Apr).

Clark, G. 1999. “Stochastic process algebra structure for insensitivity”. *In proceeding of the 7th workshop on process algebra and performance modelling*, (prensas universitarias de zaragoza, September), 63 –82.

Clarke, E.M.; E.A. Emerson; and A.P. Sistla. 1986. “Automatic Verification of Finite State Concurrent Systems Using Temporal Logic Specifications”, *in ACM transaction on programming languages and systems 8*, 244-263.

Götz, N.; U. Herzog and M. Rettelbach. 1993. “TIPP - A Language for Timed Processes and Performance Evaluation”, *The First International Workshop on Process Algebra and Performance Modelling*, (UK, University of Edinburgh, May).

Hillston J. and M. Ribaudo. 1996. “Stochastic Process Algebras: a New Approach to Performance Modeling”, *State of the Art Modelling and Simulation of Advanced Computer Systems*, Chapter 10.

HOARE, C.A.R. 1985. Communicating Sequential Processes, Prentice Hall International, London.

Milner, R. 1989. Communication and Concurrency, Prentice-Hall.

AUTHOR BIOGRAPHIES

OSMAN SALEM received his Diploma in networks and telecommunications in 2002 at ENSEEIHT of Toulouse. He is currently a Phd student in IRIT. His research interests areas are formal specification using stochastic process algebras and quality of service aspects in mobile Ad Hoc Networks. His e-mail address is: osman@irit.fr and his Web-page can be found at http://www.irit.fr/~Osman.Salem.

ABDELMALEK BENZEKRI received his Doctorat in computer science in 1989 at the Université Paul Sabatier of Toulouse. In 1998, he obtained the French Habilitation to direct research in the field of qualitative and quantitative evaluation of distributed software, at the same university. He got a full position of professor in 1999. His main research activities focused on distributed systems and on internetworking solutions. From the different EC funded projects he participated in, as the scientific person in charge of within UPS, he gained experience in managing more than 9 PhD students (4 already graduated). He is author of more than 50 papers in international journals and conferences. He was member of the French Real-time community before being interested in security and multimedia concerns. His e-mail address is: benzekri@irit.fr and his Web-page can be found at http://www.irit.fr/~Abdelmalek.benzekri.