Functional Modelling and Performance Evaluation for Two Class Diffserv Router using Stochastic Process Algebra

Osman Salem Abdelmalek Benzekri

Institut de recherche en informatique de Toulouse,
Université Paul Sabatier,
118 Route de Narbonne - 31062 Toulouse Cedex 04 - France
Téléphone: +33 05 61 55 60 86 - Télécopie: +33 05 61 52 14 58
E-mail: {osman ,benzekri}@irit.fr

Abstract: This paper describes the use of stochastic process algebra to model and to evaluate the performance of a two class DiffServ router. This specification is done by means a set of powerful operators of Extended Markovian Process Algebra (EMPA) language, and then studied from the functional and the performance point of view.

Keywords: Stochastic Process Algebra; Markov Models; Performance Evaluation; DiffServ.

1. Introduction

Network devices can be described using classical process algebra which provides a means for constructing an abstract model of the device in question [Bernardo, 1998a]. This model is used only to establish the correct functional behaviour by deriving qualitative properties such as freedom from deadlock or livelock [Benzekri, 2002]. Performance evaluation of the model was in a separate phase, after the fully design and implementation of the model. Consequently, if the performance is detected to be poor, the model will be redesigned with negative consequences for both design cost and time lost, where the need of integrating the performance analysis into the design process.

Stochastic Process Algebra (SPA) has been developed for this purpose. It is a formal specification technique which extends classical process algebras via the inclusion of timing information by using random variables in the generated models, in order to express the durations of an activity [Benzekri, 2002; Brinksma and Hermanns, 2001]. Once the model has been defined and parameterized, it can be used to investigate numerically the performance parameters.

Designing DiffServ [Nichols and Blake, 1998] routers using SPA is a complicated task because there are many factors to be considered such as penalty policy of malicious traffic inside a class. This policy may fluctuate from delaying to dropping packets from this flow. In contrast, formal specification can be a valuable aid to routers designers as it allows a range of options for configuration to be explored in a precise setting, such as policy requirements which may be clarified during the performance evaluation of a router model, because it becomes evident what information about the state of the model is required to ensure that it operates effectively. For example, it would be possible for the designer to demonstrate that under any constraints, a minimum threshold for throughput and delay may be satisfied for such class of traffic.

This paper is organized as follows. In section 2 we recall the syntax and semantics of Extended Markovian Process Algebra EMPA, and we illustrate the difference with other existing SPA languages. In section 3 we recall the principle of the DiffServ technology. In section 4 we give the specification of a DiffServ router. In section 5 we analyze the performance of this model by using EMPA algebraic reward method and the CTMC diagram derived from model specification. Finally, conclusions work is presented.

2. Stochastic Process Algebra

Stochastic Process Algebras are formal descriptions techniques used to describe the functionality of concurrent and distributed systems and to analyze their related performance [Benzekri, 2002; Brinksma and Hermanns, 2001]. Several SPA languages have been appeared in the literature, these include PEPA [Hillston, 1996], TIPP [Herzog, 1993], EMPA [Bernardo, 1998b]. These languages have been introduced as an extension to classical process algebras like CCS [Milner, 1989] and CSP [Hoare, 1985]. They are abstract languages constructed from a small set of powerful operators where it is possible to construct algebraic models whose key features are: compositionality (which allows the designer to build a complex model from smaller ones by means of languages operators, and to study the behaviour of each component separately), and abstraction (which allows the internal details of a system description to be hidden from an external observer at analysis time). In these languages, systems are modeled as a collection of entities, called agents or processes, which execute actions. These actions are the building blocks of these languages and they are used to describe sequential behaviours which may run concurrently by synchronizations or by communications between them.

These languages propose the same approach to performance modeling: a random variable is associated with each action, representing its duration. This random variable is assumed to be exponentially distributed and this leads to a clear relationship between the process algebra model and a Continuous Time Markov Chains (CTMC). Via this underlying CTMC derived from the model semantic description [Benzekri, 2002], different types of analysis may be performed, like steady-state and transient probability distribution. This analysis is done through the compilation of the infinitesimal generator matrix of the Markov diagram.

In this paper, we will use the Extended Markovian Process Algebra (EMPA) language [Bernardo, 1998b] which is supported by a tool called TwoTowers. EMPA is inspired and developed on the basis of PEPA (Performance Evaluation Process Algebra [Hillston, 1996]) and TIPP (TImed Processes and Performability evaluation) languages [Herzog, 1993]. It extends these languages by including three different kinds of actions: exponentially timed actions, passive actions and prioritized weighted immediate actions. In addition to this reason, EMPA allows one to specify performance measures with the algebraic specifications of the system through atomic rewards attached to states and transitions of the Markov chains (MC for short). This leads to an automatic derivation of performance measures and may avoid a full scan of the CTMC diagram. The syntax of EMPA can be summarized by the following expression:

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

Since the deadlock term "0", the prefix operator "<a,λ>._", the functional abstraction operator "_/L", the functional relabeling operator "_[φ]", the alternative choice operator "_+_", the cooperation operator "_||s_" and the constant operator are the same operators used in classical process algebras. Due to lack of space, the reader is referred to [Bernardo, 1998b] for an extensive presentation of EMPA syntax and semantics.

In EMPA, every activity is represented by <a,λ> which means the execution of action "a" after exponential distributed delay with rate "λ" (denoted by $F(t)=1-e^{-\lambda t}$). An immediate action is represented with rate $\lambda = \infty$ or "$<a,\infty_{L,W}>$", where L is used to express the priority level and W is used for the probability weight. In some cases, the rate of an action is outside the control of this component, such actions are carried out jointly with another component in order to model activities waiting for synchronization, so this component is playing a passive role and is recorded by the distinguished symbol "*".

The choice in the alternative composition operator "_+_" is governed by the race policy, where the action with least duration will be executed. In this situation, immediate actions take precedence over exponentially distributed actions and over other immediate actions having small priority level with respect to their levels. If two immediate actions have the same priority level, they will be executed according to the probability associated with each one.

The main difference between EMPA and other stochastic process algebras languages (PEPA and TIPP) is related to synchronization of stochastic actions of two concurrent components in the model [Brinksma and Hermanns, 2001]. For illustrating this difference, we take the following behaviour expression:

$$(a,\lambda).B \parallel a \parallel (a,\mu).C = (a, \lambda*\mu).(B \parallel a \parallel C)$$

And we will show the solution adopted by these languages for finding the function "*" (cooperation law) that makes the equality in the previous equation hold.

In PEPA, the function "*" corresponds to the minimum of the related rates, in order to express that synchronization between these components is determined by the rate of the slowest one. In contrast TIPP adopts another solution, where "*" is interpreted as the ordinary multiplication, but the operational intuition behind the choice of multiplication for this function is not at all obvious and no useful stochastic interpretation of this solution other than algebraic simplicity. EMPA deals with synchronization by adopting the client/server (or master/slave) model, where the server determines the rate of service and the client plays a passive role with this respect like in CCS and CSP. This synchronization discipline imposes that action (a,λ) can be synchronized with action (a,μ), if and only if min(λ,μ) is unspecified and the rate of the resulting action is given by max(λ,μ), or in other words, in synchronization at most one action must be active and all the other involved actions must be passive.

3. The DiffServ Router

Differentiated services (DiffServ) [Nichols and Blake 1998] is a set of technologies which allow network service providers to offer services with different kinds of network quality of service (QoS) to different customers and their traffic streams, depending to a contract (Service Level Agreement or SLA) between them.

DiffServ work by dividing traffic into many classes by marking a field in the IP packet header, called the Differentiated Services Code Point (DSCP) field. Its value depends on the customer profile and the traffic requirement. Network elements serve these classes with different priorities with respect to the DSCP field content. Applications requiring low loss, low latency, low jitter and assured bandwidth service generally send data as expedited forwarding (EF) class packets. This class is used for loss and delay sensitive applications such as voice over IP (VoIP). Assured forwarding (AF) class offers a lower priority service from the previous one (EF), and itself is subdivided into four subclasses and each of these four classes is also divided into three subclasses (gold, silver, bronze) [Nichols and Blake 1998]. Generally AF carries best effort TCP data, such as HTTP and FTP traffic applications.

Like we have seen that a DiffServ router has a large number of classes defined, but the most essential use of DiffServ is to provide support for the two most common applications: voice and video traffic with high priority level, and best effort data (TCP) with low priority level. This is why in the rest of this paper, we will be concerned only with the modelling of such a two classes router and we will denote these classes by H (high priority level) and L (low priority level), instead of modeling all classes in order to prevent a huge number of state and the state space explosion problem when analyzing the model by existing tools.

The DiffServ router is composed from a classifier and a traffic conditioner like appears in figure.1. Traffic conditioners may contain meters, markers, droppers and shapers. We must notice that some of these blocks may be aggregated in another block or may be omitted. For example, in the case where no traffic profile is in effect, packets may only pass through a classifier and a marker, and in the case of core routers, marker may be omitted because packets were coded at the ingress router. Readers interested about the DiffServ technology can refer to [Nichols and Blake, 1998; Blake, 1998].

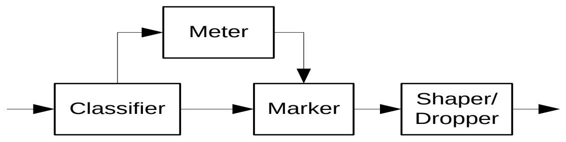

Figure 1. Logical view of a DiffServ router

4. The Specification

We take advantage from the compositional feature of SPA in order to model the DiffServ router which appears in figure 2. This feature allows us to deal with five entities: classifier, marker, meter, dropper and priority queueing.

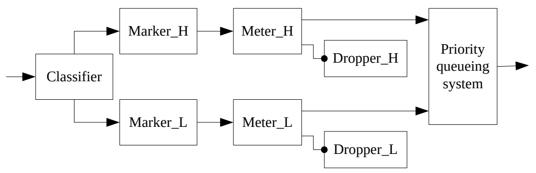

Figure 2. An ingress DiffServ router

In this specification, we suppose that the customer has a service level specification (SLS) which specifies 2 service levels, to be identified to the provider by DSCP High and DSCP Low. Each components of this DiffServ router model [Bernet and Blake, 2002] can be specified as follows:

Classifier: It takes a single traffic stream as input and generates N logically separated traffic streams as output. Classifier can be specified by a filter which has one input and N outputs. Figure 3 show a classifier that separates input traffic into one of two output streams based on matching filters:

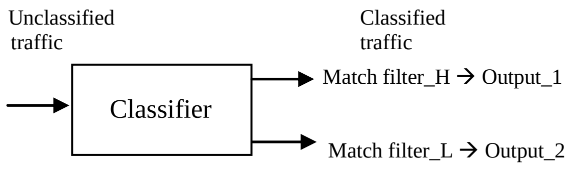

Figure 3. Classifier

The specification of this agent using EMPA is:

Classifier = <packet_arrival,λ>.<check_pckt_header,θ>.
    <classify,∞_{11}>.(<send_message$_H$,λ_1>.Classifier
  + <send_message$_L$,λ_2>.Classifier);

Marker: The marker sets the DS field of each packet received from the classifier to a particular code-point (e.g., DSCP). It can be represented logically by a box with one input one output like appear in figure 4.

Figure 4. Marker

In our model, two markers are needed, one for the high priority level traffic and the other for the low priority level traffic. Their specification is the following:

Marker_H = <send_message$_H$,*>.<mark_DSCP$_H$,α>.
    <send_to_meter$_H$,∞_{11}>.Marker_H;

Marker_L = <send_message$_L$,*>.<mark_DSCP$_L$,α>.
    <send_to_meter$_L$,∞_{11}>.Marker_L;

Meter: It is used to monitor the traffic stream and sends malicious packets to the dropper agent, in order to prevent high level traffic from monopolizing the network resource. Figure 5 illustrates a simple meter with two levels of conformance. It will measure the rate of each traffic to determine its conformance. So, if the packet is judged conformed, then it will be sent to the priority queueing system in order to be served (forwarded to next hop), else the packet will be sent to the dropper. This agent can be specified by the following:

Meter0_H= <send_to_meter$_H$,*>.<start_timer,∞_{11}>.
    <arr$_H$,λ_H>.Meter1_H;

Meter1_H= <send_to_meter$_H$,*>.<get_current_time,∞_{11}>.
    <compute_elapsed_time,∞_{11}>.(<time_elapsed,∞_{11}>.
    <average_conform,∞_{11}>.<reset_timer,∞_{11}>.
    <arr$_H$,λ_H>.Meter1_H
  + <time_not_elapsed,∞_{11}>.<avrg_not_conform,∞_{11}>.
    <send_to_dropper$_H$,γ>.Dropper_H
  + <timeout,η>. Meter0_H);

Meter0_L = <send_to_meter$_L$,*>.<start_timer,∞_{11}>.
<arr$_L$,λ_L>.Meter1_L;
Meter1_L= <send_to_meter$_L$,* >.<get_current_time,∞_{11}>.
<compute_elapsed_time,∞_{11}>.(<time_elapsed,∞_{11}>.
<average_conform,∞_{11}>.<reset_timer, ∞_{11}>.
<arr$_L$,λ_L>.Meter1_L
+ <time_not_elapsed,∞_{11}>.<avrg_not_conform,∞_{11}>.
<send_to_dropper$_L$,γ>.Dropper_L
+ <timeout,η>.Meter0_L);

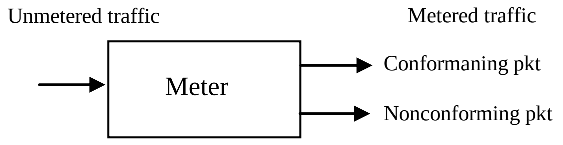

Figure 4. Meter

Dropper: The dropper discards some or all malicious packets in a traffic stream according to the service provider policy. The dropper can be implemented as a special case of shaper by setting the buffer size to zero. It can be represented logically by a box with one input one output and its specification is the following:
Dropper_H = <send_to_dropper,*>.<discard,∞_{11}>.
Meter1_H
Dropper_L = <send_to_dropper,*>.<discard,∞_{11}>.
Meter1_L
Priority queueing: The final agent is the queue where packets wait before being served (forwarded to next hop). We have taken a preemptive queue (e.g., arriving of high priority packet will interrupt the service of low priority packet already in service phase) with priority inter-arrival policy in the sense that if the queue is full and a high priority packet arrives, it will drop a low priority packet (if the queue contains at least one) in order to accept the high priority packet, but if it contains only a high priority packet, the arrival packet will be lost (dropped). The specification of this queue (M/M/1/N) [Thomas and Hillston, 1997] is the following:
Queue$_{0,0}$ = <arr$_H$,*>. Queue$_{1,0}$ + <arr$_L$,*>. Queue$_{0,1}$;
Queue$_{i,0}$ = <arr$_H$,*>. Queue$_{i+1,0}$ + <arr$_L$,*>. Queue$_{i,1}$
+ <deliver$_H$,*>. Queue$_{i-1,0}$; (if 0<i<N-1)
Queue$_{0,j}$ = <arr$_H$,*>. Queue$_{1,j}$ + <arr$_L$,*>. Queue$_{0,j+1}$
+ <deliver$_L$,*>. Queue$_{0,j-1}$; (if 0<j<N-1)
Queue$_{i,j}$ = <arr$_H$,*>. Queue$_{i+1,j}$ + <arr$_L$,*>. Queue$_{i,j+1}$
+ <deliver$_H$,*>.Queue$_{i-1,j}$
+ <deliver$_L$,*>.Queue$_{i,j-1}$; (if i,j>0 and i+j<N-1)
Queue$_{N-1,0}$ = <deliver$_H$,*>. Queue$_{N-2,0}$;
Queue$_{0,N-1}$ = <deliver$_L$,*>.Queue$_{0,N-2}$
+ <arr$_H$,*>.<loose$_L$,$\infty_{2,1}$>.Queue$_{1,N-2}$;
Queue$_{i,j}$ = <deliver$_H$,*>.Queue$_{i-1,j}$ + <deliver$_L$,*>.Queue$_{i,j-1}$
+ <arr$_H$,*>.<loose$_L$,$\infty_{2,1}$>.Queue$_{i+1,j-1}$;
(if i,j>0 and i+j = N-1)
Pre_Server = <deliver$_H$,$\infty_{2,1}$>.<serve,μ>.Pre_Server
+ <deliver$_L$,$\infty_{1,1}$>.ServePLLow;
ServePLLow = <serve,μ>.Pre_Server
+ <deliver$_H$, $\infty_{2,1}$>.<serve,μ>. ServePLLow;

In order to obtain the complete DiffServ router specification, the individual agents described above need to be composed like in the following expression:
DiffServ = Classifier $\|_T$ Markers $\|_S$ Meters $\|_M$ Droppers
$\|_{Arr}$ Queue00 $\|_{Del}$ Pre_emp_Server
Markers = Marker_L || Marker_H
Meters = Meter0_H || Meter0_L
Droppers = Dropper_L || Dropper_H
T = {send_msg$_H$, send_msg$_L$};
S = {send_to_meter$_H$, send_to_meter$_L$};
M = {send_to_dropper$_H$, send_to_dropper$_L$};
Arr = {arr$_H$, arr$_L$};
Del = {deliver$_H$, deliver$_L$};
When the model is loaded in TwoTowers [Bernardo, 1998b], its descriptions are syntactically and semantically analyzed using a parser for detecting errors, then TwoTowers will find all possible states and transitions or in another expression the labelled transition diagram.

5. Performance Analysis

TwoTowers will give us the steady state and the transient state distribution probability vector. So given the CTMC diagram and the value of the probability distribution in steady and transient states, we can evaluate the performance of the system by using queueing system theory. For example, the throughput which is given by the service rate multiplied by the stationary probability of being in a state where service action can be provided is given by the following formula:

$$T = \sum_{i=2}^{N} \mu.\pi_i$$

The utilization rate was defined to be the percentage of time the router spent in doing useful work by the fraction of time, and which is the sum of the stationary probabilities of states where there is at least one packet in the system. It is given by the following formula:

$$U = \sum_{i=2}^{N} (1.\pi_i) = \sum_{i=2}^{N} \pi_i$$

The vector distribution probability π_i for all states is given by TwoTowers and the utilisation rate can be calculated by a simple addition.
The Markovian analyzer implemented in TwoTowers allow an automatic derivation of performance model and may allow us to avoid the full scan to the CTMC diagram, which will be exceedingly expensive, especially if we have a large number of states. The performance aspects of a system model in EMPA, can be taken into account in the early stages of its design (with algebra description), where performance measures can be specified by attaching a yield reward y_i to every state i, which expresses the rate at which reward is accumulated at state i, and by attaching a bonus reward $b_{i,j}$ to every transition from state i to state j, which expresses the instantaneous

gain due to the execution of the transition from state i to state j. Readers interested about yield and bonus rewards can refer to [Bernardo, 1997]. Actions with reward will be specified in EMPA according to the following syntax:

A= <a,r,y,b>.A

Given yield and bonus rewards, the corresponding stationary performance measure can be computed in EMPA according to the following formula:

$$\sum_{i=1}^{N} y_i .\pi_i + \sum_{i=1}^{N}\sum_{j=1}^{N} b_{ij}.\pi_i .q_{ij}$$

Many performance measures can be obtained using this formula, for example: if we want to compute the throughput, we must replace every action of the form <serve,μ> with <serve,μ,μ,0> (e.g., y_i=μ and b_{ij}=0) for obtaining the following equation:

$$\sum_{i=1}^{N} y_i .\pi_i + \sum_{i=1}^{N}\sum_{j=1}^{N} b_{ij}.\pi_i .q_{ij} = \sum_{i=2}^{N} \mu.\pi_i$$

EMPA will take into account all states that provide the action "serve" and will assign a reward to them. The first state is where no packet in the router and this is why it can not provide the action "serve".

As a performance measure, we have computed the throughput and the router utilization by using the reward technique of EMPA. This is done by replacing every action <serve,μ> by <serve,μ ,μ,0> in our semantic model for obtaining the throughput, and by replacing every action <serve,μ> by <serve,μ,1,0> in order to obtain the utilization rate. In contrast, the algebra based method (reward technique) fails to determine the mean number and the mean waiting time of packets for each class due to the additivity assumption of transition labeled with "serve" action, and values for these performance aspects were calculated by a manual full scan to the transformed specification (CTMC diagram) and by using the probability distribution vector given by TwoTowers.

The mean number of packet in the system can be obtained by using the following formula:

$$\text{The mean number of packet} = \sum_{i=2}^{N} i.\pi_i$$

And the mean packet delay (MPD) for each class is found by using Little's law:

$$\text{MPD} = \frac{\text{Mean system size}}{\text{Mean packet arrival rate}}$$

$$\text{Mean packet arrival rate} = \sum_{i=1}^{N} \lambda_i.\pi_i$$

Where λ_i take the value of λ_H for packet with high priority and λ_L for packet with low priority.

The throughput was 2.38 packet/s, the utilisation rate was 33.34% and the mean waiting time was 1.12s for a packet in class high and 2.48s for packet in class low. These unacceptable results lead us to a set of experiment in order to detect the effect of each component at its performance. We begin by examine the effect of changing the rate (speed) of the marker at the system performance.

Figure 5.a shows that the throughput increases by increasing marker speed but reaches a threshold afterwere there is no effect of increasing its speed at the system performance, and this effect can be explained by the limited speed of the classifier. However the utilisation rate of the marker decreases significantly by the fact of speeding the marker, because packets will spend a less time before being forwarded to next stage. The mean packets waiting time decreases slightly when we decrease the rate of this component because packets spend less time in this component.

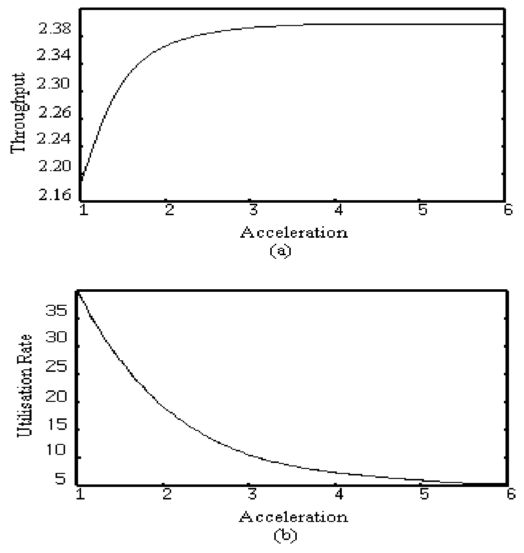

Figure 5. Effect of speeding up the marker

We have experimented the effect of speeding up the meter (increasing its rate) at the performance model. Like expected the throughput and the utilisation rate of the model increase and reach quickly a maximum threshold value (curves variation are similar to that in Figure 5).

The same experiment was repeated for the classifier and the queueing server. We have found that there was little profit from speeding up any of these components apart at the model performance.

The result of these experiments motivate us to another set of experiments in order to investigate the effect of speeding up many components at the system performance, because every time we have increased the rate of a component we have found: the throughput of the system increases, the utilisation rate fluctuate, and the mean packet delay decreases for each class but still inside a specific margin.

Figure 6 shows the result obtained by speeding up the marker and the queueing server, it can be seen from this figure that the increase of throughput is not at the expense of utilisation rate like we have seen when speeding up one component alone.

We discover from these experiments that we can use relatively a slow classifier with no big influence at the system throughput in contrast like it have been thought. This result can be interpreted by the time that packets spend in other components especially in the server queueing.

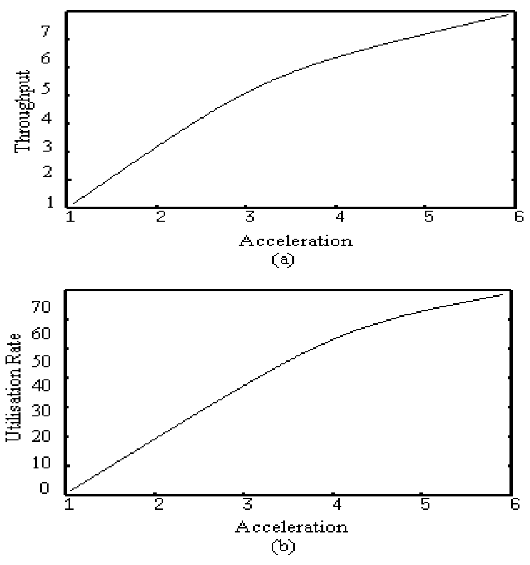

Figure 5. Effect of speeding up the marker and the server

These experiments lead us to discover some interesting information about which component limited the throughput and other performance aspect. It demonstrates that we can use a relatively slow server router (a server queue able to find the next hop) which is the most expensive component, with no significant difference in the throughput in contrast to that it has been thought originally.

6. Conclusions

The main aim of this work was to present a simple DiffServ router model and to analyze its functional and performance properties using stochastic process algebra. In order to achieve that, we begin by an algebraic description of this router using stochastic process algebra then we use EMPA tool (TwoTowers) for analyzing and detecting a missbehavioural functional error such as freedom from deadlock. After qualitative verification, a set of experiment has been done in order to detect the effect of each component at the performance of this model. Fortunately, the reward-based method in EMPA provides an automatic derivation for some performance aspect in our model (like throughput, utilisation rate...), but unfortunately not all. Therefore, a full scan to the CTMC diagram derived from the EMPA supported tool is necessary.
The coexistence of three kinds of actions in EMPA and especially the prioritized weighted action was a great characteristic because these actions are not taken in account in the performance semantic model (CTMC diagram) and this aids us to include only actions which are important for determining performance aspects.

References

Blake S. 1998, "An Architecture for Differentiated Services", RFC 2475.

Benzekri A. 2002, "Qualitative and Quantitative Evaluation using Process Algebra", The 17th International Symposium on Computer and Information Sciences, Orlando, Florida USA, Pp415-418.

Bernardo M. 1997, "An Algebra Based Method to Associate Rewards with EMPA Terms", in Proc. of the 24th Int. Coll. on Automata, Languages and Programming (ICALP), P.Degano, Lecture Notes in Computer Science, Bologna, Pp358-368.

Bernardo M. 1998a, "A Formal Approach to the Integration of Performance Aspects in the Modeling and Analysis of Concurrent Systems". International Journal of Information and Computation, Pp83-154.

Bernardo M. and Gorrieri R. 1998b, "A Tutorial on EMPA: A Theory of Concurrent Processes with Nondeterminism, Priorities, Probabilities and Time", Theoretical Computer Science, Pp1-54.

Brinksma Ed and Hermanns Holger 2001, "Process Algebra and Markov Chains", Lecture on Formal Methods and Performance Analysis, Nijmegen, Pp183–231.

Bernet Y. and Blake S. 2002, A. Smith: "An Informal Management Model for DiffServ Routers", RFC 3290.

Herzog U. 1993, "TIPP: A Language for Timed Processes and Performance Evaluation", Proceedings of the First International Workshop on Process Algebra and Performance Modelling, University of Edinburgh, UK.

Hillston J. 1996, "A Compositional Approach to Performance Modelling", Cambridge University Press.

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

HOARE C.A.R 1985, "Communicating Sequential Processes", Prentice-Hall.

Nichols K. and Blake S. 1998, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474.

Thomas N. and Hillston J. 1997, "Using Markovian Process Algebra to Specify Interactions in Queueing Systems", Technical Report, University of Edinburgh, Pp151-164.