Active Nodes Performance Analysis using PEPA

J. Hillston[1], L. Kloul[1*] and A. Mokhtari[2]

[1]LFCS, University of Edinburgh, Kings Buildings,
Edinburgh EH9 3JZ, Scotland
{jeh,leila}@inf.ed.ac.uk
[2]PRiSM, Université de Versailles, 45, Av. des Etats-Unis,
78035 Versailles Cedex, France
amok@prism.uvsq.fr

Abstract. We investigate the impact of introducing active networking on traditional packets flowing through the active nodes of a network. Using the process algebra formalism PEPA, we compare the performance of an active node and of a traditional one. We are mainly interested in performance measures such as loss rates of standard packets and the node latency for this kind of packets.

1 Introduction

In an active network the processing to be performed in its different nodes can be customized according to the user and/or application requirements. User or application specific functionalities are embedded within either the nodes of the network (programmable switches) or the packets flowing through them (capsules), in the form of methods or small programs. In the first approach, the user sends first his program packets into selected network nodes and when a user data packet arrives at these nodes, its header is evaluated and the appropriate program is executed on the packet's contents. In the second approach, each packet or capsule consists of a small program which is transmitted in-band and executed at each node along the packet path.

Although these approaches seem to be different, the concept of active networking behind them is fundamentally the same. In both approaches, the packet's content is extracted and dispatched to an environment to be executed immediately (capsules approach) or at the right moment (programmable switches approach). This suggests that the mechanisms and the primitives that are involved in these operations are not only independent of the approach used, but also of the application itself.

The need to develop a common programming model—including common models for network programs, encoding the built-in primitives available at each node and the allocation of the node resources—has shown the necessity of a common architecture able to accept different packet languages and execution

* On leave from PRISM, Université de Versailles, 45, Av. des Etats-Unis 78000 Versailles, France.

environments. The idea of an architecture based principally on three layers has been suggested in [2] and since used in several different works [4][6][7]. These layers are the *active applications layer* which specifies the application services for the users data and the network control, the *execution environments layer* which interprets the active packets and executes the active applications, and finally the *operating system* which manages several types of execution environments and the allocation of the node resources to these environments.

In this paper, we investigate the performance of active nodes in the context of the active network framework presented in [1]. This framework is based on the notion of active rules commonly used in the active database area. In an active network, we expect that the processing to be performed in the nodes can be customized according to the user or the application requirements. Active rules provide an explicit semantics which facilitates the reasoning based on the application's behaviour and its execution tuning. They allow us to describe a system or application behaviour with fine granularity which can easily evolve according to the application evolution. An application may be described as a set of active rules, where each rule is defined as an Event-Condition-Action (ECA) statement. The execution of the application consists then of event detection, condition evaluation and action launching.

Using the formalism PEPA, we compare the performance of a "passive" or standard node whose only functionality is to forward the packets flowing through it, with the performance of an active node. We are mainly interested in performance measures such as the loss rates of the standard packets and the node latency, in particular for this kind of packets.

Structure of this paper: In Section 2 we give a brief overview of the PEPA formalism. Section 3 is dedicated to the presentation of the active-rules based framework for active networks. In Section 4 we describe our PEPA model for an active-rule based node. In Section 5 we define the performance criteria we are interested in and in Section 6 we discuss the results obtained. Finally conclusions and possible extensions of this work are given in Section 7.

2 PEPA

PEPA (Performance Evaluation Process Algebra) extends classical process algebra by associating a random variable, representing duration, with every action. These random variables are assumed to be exponentially distributed and this leads to a clear relationship between the process algebra model and a continuous time Markov process. Via this underlying Markov process performance measures can be extracted from the model.

PEPA models are described as interactions of *components*. Each component can perform a set of actions: an action $a \in \mathcal{A}ct$ is described as a pair (α, r), where $\alpha \in \mathcal{A}$ is the *type* of the action and $r \in \mathbb{R}^+$ is the parameter of the negative exponential distribution governing its duration. Whenever a process P can perform an action, an instance of a given probability distribution is sampled: the resulting number specifies how long it will take to *complete* the action. A small but powerful set of combinators is used to build up complex behaviour from simpler behaviour. The combinators are familiar from classical process algebra: prefix, choice, parallel composition and abstraction. We explain each of the combinators informally below. A formal operational semantics for PEPA is available in [5].

Prefix: A component may have purely sequential behaviour, repeatedly undertaking one action after another and eventually returning to the beginning of its behaviour. In this case the prefix combinator "." is used to designate the first action, e.g. $(\alpha, r).P$ will carry out an activity of type α with an average duration of $1/r$ and then behave as component P.

In some cases, the rate of an action is outside the control of this component. Such actions are carried out jointly with another component, with this component playing a passive role. In this case the rate of the activity is denoted by the distinguished symbol, $\top$ (called "top").

Choice: A choice between two possible behaviours is represented as the sum of the possibilities, e.g. $(\alpha, r).P + (\beta, s).Q$. A race condition is assumed to govern the behaviour of simultaneously enabled actions so the choice combinator represents pre-emptive selection with re-sampling. The continuous nature of the probability distributions ensures that the actions cannot occur simultaneously. Thus a sum will behave as one of its summands. When an action has more than one possible outcome it is represented by a choice of separate actions, one for each possible outcome. The rates of these actions are chosen to reflect their relative probabilities.

Parallel composition: Parallel composition is used to represent the cases when we expect two components of the system to work together to achieve some action. This will require them to *cooperate*. For example, the system $P \bowtie_{L} Q$ consists of two components P and Q which must cooperate to achieve actions which are in the set L. Actions with types not in this set may be carried out independently and concurrently by the two components. L is termed the *cooperation set*. Actions in this set require the simultaneous involvement of both components. The resulting action, a *shared* action, will have the same type as the two contributing actions and a rate reflecting the rate of the action in the slowest participating component. Note that this means that the rate of a passive action will become the rate of the action it cooperates with.

Abstraction: It is often convenient to hide some actions, making them private to the component or components involved. The duration of the actions is unaffected, but their type becomes hidden, appearing instead as the unknown type τ. Components cannot synchronise on τ.

Use of the hiding combinator has two implications. Firstly, it ensures that no components added to the model at a later stage can interact, or interfere, with this action. Secondly, private actions are deemed to have no contribution to the performance measures being calculated and this might subsequently suggest simplifications to the model.

Using constants to name components and recursive definitions we are able to describe components with infinite behaviours without the use of an explicit recursion operator.

3 The Active Rules Based Framework

The idea behind an active network consists of providing triggering facilities which allow some nodes (active nodes) to react to external events conveyed by information packets or generated by the operating system environment of the active node. An application may be described as a set of active rules, where each rule is defined as an Event-Condition-Action statement.

3.1 Active Rule Components

The general form of an active rule is the following:

On <event type name> *If* <condition expression> *Then* <action invocation>.

- *Event Specification:* An event is any discrete signal that necessitates a reaction from an active node by triggering an application service. Examples of events are the arrival of a packet to a node, the ACK, NACK or the ACK-Timeout that a node may receive.
- *Condition Specification:* A condition may be a logical formula or a boolean function. Variables in the condition part may refer to the data in the event instance or to persistent data in the node or both. For example, a simple condition may test whether a packet has been sent or not, whether it is still in the cache or not, etc.
- *Action Specification:* A rule action may be any action the active node can perform for the purpose of data communication, network management, or application service providing. Examples: send a packet, send an ACK or a NACK, make a resource allocation, apply a network reconfiguration procedure, etc.

Rules are organised into modules and an application service is defined by one or several modules. The header of each module provides the set of semantic parameters that govern the behaviour of the rules (see [1] for more detail). According to these parameters, the execution of an application consists in detecting the events, evaluating the condition and finally launching the corresponding actions.

3.2 Active Node Architecture

Each node of the network is defined by a layered architecture[1] as shown in Figure 1. Besides the operating system, it is composed of three layers: the *application services*, the *active monitors* and the *virtual machine.*

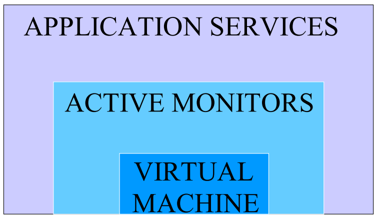


Fig. 1. Active Node Architecture

- *The Active Monitor* or execution environment implements the operational semantics of a class of application programs, each defined by a set of ECA rules. An active node is equipped with one or several active monitors, each devoted to a class of applications requiring the same execution semantics. Active monitors are dynamically loaded (published) in one or several active nodes.
- *The Application Service* is defined as a set of active rules which will be executed with respect to a specific active monitor. As for active monitors, application services are dynamically loaded in the active nodes. The binding between an application service and the corresponding active monitor is done when the application service reaches an active node.
- *The Virtual Machine* is the layer between the node operating system and the active monitors. It is implemented once for all and provides the minimal functionalities of:
  - routing packets (send, receive, annotate packets),
  - analysing packets to differentiate between those corresponding to active monitors (*m-packets*), those corresponding to application services (*s-packets*) and those corresponding to regular data (*d-packets*).
  - bootstrapping the active monitor packets in order to make the node active,
  - loading and binding application services to be executed in the active node,
  - binding application services to active monitors in active nodes or application data to application services.

Modelling the active node architecture of Figure 1 and its execution mode assumes that the node is viewed as a set of finite queues as depicted in Figure 2. The packets arriving at the node are queued in $queue_{in}$, before being dispatched according to their type as follows:

- the passive packets (to forward) are directly shortcut to the outgoing queue, called $queue_{out}$,
- the active data packets (*d-packets*) are oriented to the events queue denoted $queue_{event}$,

– the application service packets (*s-packets*) are oriented to the service queue denoted $queue_s$,
– the active monitor packets (*m-packets*) are oriented to the monitors queue noted $queue_m$.

Besides the *d-packets*, $queue_{event}$ receives also internal events such as the events generated by the system and the application services. If a *d-packet* in service in $queue_{event}$ references a non-existent active code packet (*s-packet* or *m-packet*) in the node, the packet is requeued in $queue_{event}$ and a request for the missing packet(s) is sent to the network using the output queue. Moreover the execution of an active application can result in the generation of new active packets sent to the outgoing queue.

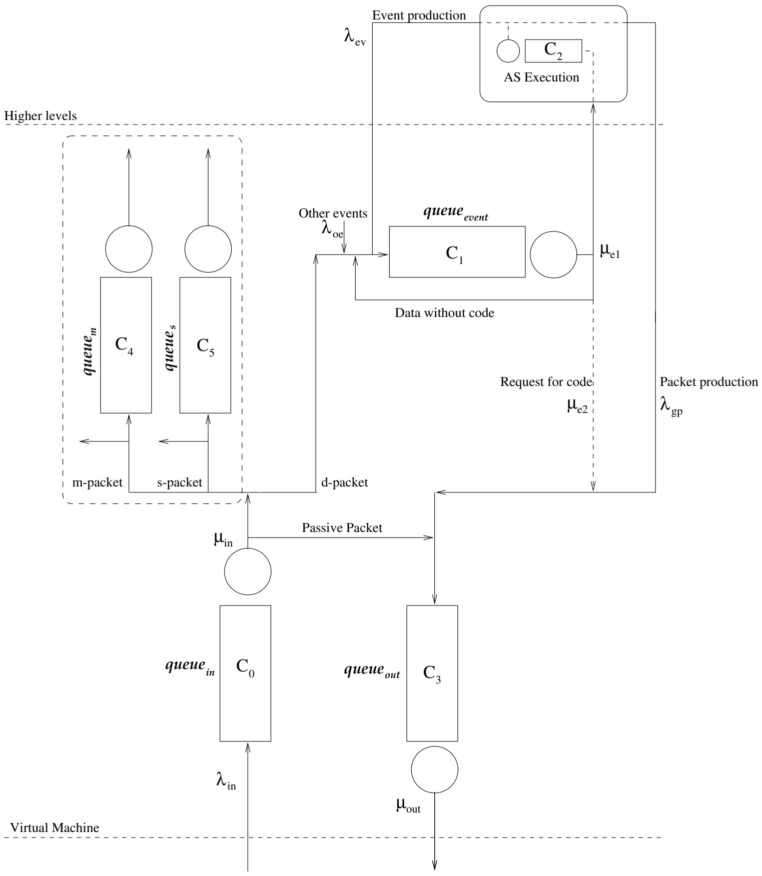


Fig. 2. Queueing network modelling the Active Node

4 The PEPA Model

As we are interested in the impact of the active networking on the traditional packets (passive packets), in terms of loss rates and node latency, our PEPA model concerns only the input, the output, the event and the application service queues. Therefore, the PEPA model is composed of four components $Buffer_0^{(in)}$, $Buffer_0^{(out)}$, $Buffer_0^{(event)}$, and $Buffer_0^{(as)}$ respectively. Let C_j, $j = 0, \ldots, 3$ be the buffer capacity of respectively $queue_{in}$,$queue_{event}$,$queue_{as}$ and $queue_{out}$.

Component $Buffer_0^{(in)}$**:** The arrivals from the network to the input queue of the node are modelled using action type in. As we distinguish between the packet types, the service of a *d-packet*, an *s-packet*, an *m-packet* and a passive packet is modelled using action type $service_d$, $service_s$, $service_m$ and $service_p$ respectively, q_x being the probability to serve a packet of type $x = d, s, m, p$. The complete behaviour of $Buffer_0^{(in)}$ is as follows.

$$
\begin{array}{ll}
Buffer_0^{(in)} & \stackrel{def}{=} (in, \lambda_{in}).Buffer_1^{(in)} \\
Buffer_1^{(in)} & \stackrel{def}{=} (in, \lambda_{in}).Buffer_2^{(in)} + (service_d, q_d \times \mu_{in}).Buffer_0^{(in)} \\
 & \quad +(service_p, q_p \times \mu_{in}).Buffer_0^{(in)} + (service_s, q_s \times \mu_{in}).Buffer_0^{(in)} \\
 & \quad +(service_m, q_m \times \mu_{in}).Buffer_0^{(in)} \\
\vdots & \quad \vdots \\
Buffer_{C_0}^{(in)} & \stackrel{def}{=} (in, \lambda_{in}).Buffer_{C_0}^{(in)} + (service_d, q_d \times \mu_{in}).Buffer_{C_0-1}^{(in)} \\
 & \quad +(service_p, q_p \times \mu_{in}).Buffer_{C_0-1}^{(in)} + (service_s, q_s \times \mu_{in}).Buffer_{C_0-1}^{(in)} \\
 & \quad +(service_m, q_m \times \mu_{in}).Buffer_{C_0-1}^{(in)}
\end{array}
$$

Component $Buffer_0^{(event)}$**:** The arrival of internal events to this queue are modelled using action type in_{oe}. The other arrivals are the events resulting from the execution of application services, modelled using action type $generate_{ev}$ and the *d-packets* after a service in the input queue ($service_d$). We use $service_{e2}$ to model a service of a *d-packet* which references a non-existent active code packet. Once this packet is detected, it is requeued and a request for the corresponding active code is generated and sent to the output queue. The other *d-packets* are served ($service_{e1}$) and sent to the application service queue.

$$
\begin{array}{ll}
Buffer_0^{(event)} & \stackrel{def}{=} (service_d, \top).Buffer_1^{(event)} + (in_{oe}, \lambda_{oe}).Buffer_1^{(event)} \\
 & \quad +(generate_{ev}, \top).Buffer_1^{(event)} \\
Buffer_1^{(event)} & \stackrel{def}{=} (service_d, \top).Buffer_2^{(event)} + (in_{oe}, \lambda_{oe}).Buffer_2^{(event)} \\
 & \quad +(generate_{ev}, \top).Buffer_2^{(event)} + (service_{e1}, q \times \mu_{e1}).Buffer_0^{(event)} \\
 & \quad +(service_{e2}, (1-q) \times \mu_{e2}).(generate_{req}, \mu_{e2}).Buffer_1^{(event)} \\
\vdots & \quad \vdots \\
Buffer_{C_1}^{(event)} & \stackrel{def}{=} (service_d, \top).Buffer_{C_1}^{(event)} + (in_{oe}, \lambda_{oe}).Buffer_{C_1}^{(event)} \\
 & \quad +(generate_{ev}, \top).Buffer_{C_1}^{(event)} + (service_{e1}, q \times \mu_{e1}).Buffer_{C_1-1}^{(event)} \\
 & \quad +(service_{e2}, (1-q) \times \mu_{e2}).(generate_{req}, \mu_{e2}).Buffer_{C_1}^{(event)}
\end{array}
$$

Component $Buffer_0^{(as)}$: To capture the behaviour of application services execution queue, we need to use three actions types: $service_{e1}$ describes the arrival of an event which triggers the execution of an application service. This execution may result in the generation of both new events ($generate_{ev}$) with rate λ_{ev} and new packets ($generate_p$) with rate λ_{gp}. The complete behaviour of $Buffer_0^{(as)}$ is as follows:

$$
\begin{array}{ll}
Buffer_0^{(as)} & \stackrel{def}{=} (service_{e1}, \top).Buffer_1^{(as)} \\
Buffer_1^{(as)} & \stackrel{def}{=} (service_{e1}, \top).Buffer_2^{(as)} + (generate_{ev}, \lambda_{ev}).Buffer_0^{(as)} \\
 & \quad +(generate_p, \lambda_{gp}).Buffer_0^{(as)} \\
\vdots & \quad \vdots \\
Buffer_{C_2}^{(as)} & \stackrel{def}{=} (service_{e1}, \top).Buffer_{C_2}^{(as)} + (generate_{ev}, \lambda_{ev}).Buffer_{C_2-1}^{(as)} \\
 & \quad +(generate_p, \lambda_{gp}).Buffer_{C_2-1}^{(as)}
\end{array}
$$

Component $Buffer_0^{(out)}$: The packets arriving at the output queue are either passive packets after a service at the input queue ($service_p$), or requests for active code generated by the events queue ($generate_{req}$) or packets resulting from the execution of an application service ($generate_p$). Activity $service_{out}$ models the service in this queue. The complete behaviour of this component is as follows.

$$
\begin{array}{ll}
Buffer_0^{(out)} & \stackrel{def}{=} (generate_p, \top).Buffer_1^{(out)} + (generate_{req}, \top).Buffer_1^{(out)} \\
 & \quad +(service_p, \top).Buffer_1^{(out)} \\
Buffer_1^{(out)} & \stackrel{def}{=} (generate_p, \top).Buffer_2^{(out)} + (generate_{req}, \top).Buffer_2^{(out)} \\
 & \quad +(service_p, \top).Buffer_2^{(out)} + (service_{out}, \mu_{out}).Buffer_0^{(out)} \\
\vdots & \quad \vdots \\
Buffer_{C_3}^{(out)} & \stackrel{def}{=} (generate_p, \top).Buffer_{C_3}^{(out)} + (generate_{req}, \top).Buffer_{C_3}^{(out)} \\
 & \quad +(service_p, \top).Buffer_{C_3}^{(out)} + (service_{out}, \mu_{out}).Buffer_{C_3-1}^{(out)}
\end{array}
$$

The complete system: The complete behaviour of the system is given by the following equation:

$$
System \stackrel{def}{=} \left(\left(Buffer_0^{(in)} \underset{\mathcal{K}}{\bowtie} Buffer_0^{(event)} \right) \underset{\mathcal{L}}{\bowtie} Buffer_0^{(as)} \right) \underset{\mathcal{M}}{\bowtie} Buffer_0^{(out)}
$$

where the cooperation sets are defined as $\mathcal{M} = \{service_p, generate_p, generate_{req}\}$, $\mathcal{L} = \{service_{e1}, generate_{ev}\}$ and $\mathcal{K} = \{service_d\}$.

5 Performance Evaluation Criteria

The performance measures of the system we are interested in are the throughput, the loss probability of the standard or traditional packets and the node latency in particular for this type of packets. The loss probability is computed at both levels, the input and the ouput queues. At the former level, we consider the states where activity *in* fails, states which correspond to a full input queue. At the latter level, we consider the states where activity $service_p$ may fail, states which correspond to a full output queue.

Similarly the latency of the active node for the standard packets is the latency experienced by these packets at both the input and the output queues. To compute this performance criterion, both queues are considered as $M/M/1/C$ queues and the well known formula of the mean response time is used.

In the following, we present the numerical results obtained for these rewards after solving the model using the PEPA Workbench [3].

6 Numerical Results

To compute the two performance criteria we have defined above, we need to define the input parameters of the model. First consider the external arrivals of packets to the node. As these arrivals are composed of active packets and standard packets, we assume in our experiments that $\lambda_{in} = \lambda_{in,a} + \lambda_{in,s}$, where $\lambda_{in,a}$ is the arrival rate of active packets and $\lambda_{in,s}$ the arrival rate of standard packets. As we are interested in the performance of the node with respect to the traditional packets, we define $\lambda_{in,s} = K \times \lambda_{in}$ where K is the proportion of standard packets which varies between 0.2 and 0.8. According to the value of K, the proportion of active packets is divided between the different types of active packets as defined in Table 1. We assume that all buffers have the same capacity $C_i = 5$, $i = 0, \ldots, 3$. The resulting Markov chain has 1584 states and 12444 tansitions. The other parameters of the model are summarised in Table 1.

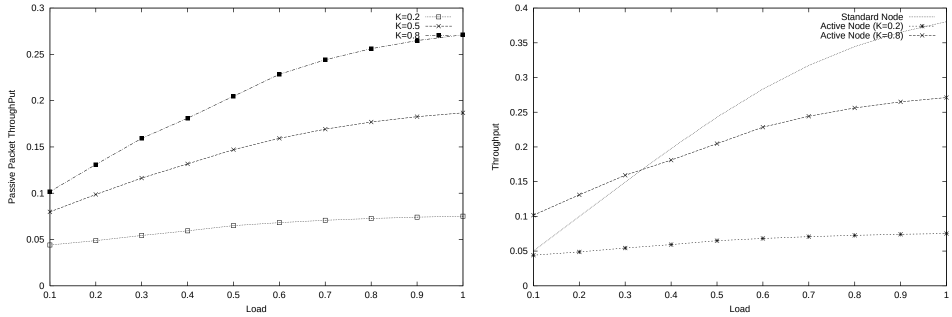
Active Packets Repartition	Arrival Rates	Service Rates	Probabilities
d_packet: 75%	$\lambda_{ev} = 0.1$	$\mu_{in} = 0.5$	$q_p = K$
m_packet: 12.5%	$\lambda_{oe} = 0.1$	$\mu_{out} = 0.5$	$q = 0.5$
s_packet: 12.5%	$\lambda_{gp} = 0.1$	$\mu_{e_1} = 0.5$; $\mu_{e_2} = 0.2$	

Table 1. Input parameters

The results obtained are depicted in Fig. 3-5. All curves are plotted versus the load of the input queue; we decrease the inter-arrival delay of packets ($1/\lambda_{in}$), increasing thus the load of the node.

1. In the first part of our experiments, we are interested in the throughput of the active node and the impact of the active packets on the throughput of the standard or passive packets. Fig. 3 summarises the main results we have obtained.

- Fig. 3 (a) depicts the throughput of the node for passive packets. Obviously this throughput increases as the proportion of passive packets K increases and also as the arrival rate to the input queue λ_{in} increases. Note that when the input file begins to be heavily loaded the throughput tends to be stable and this for all values of K.



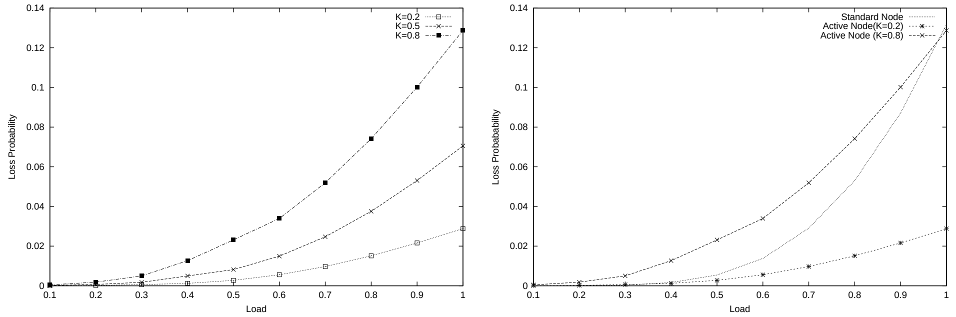
(a) Passive packet throughput *(b) Nodes throughput*

Fig. 3. The throughput

- To have an idea about the impact of providing standard nodes with new functionalities on passive packets throughput, we compare the throughput of the active node for passive packets considering $K = 0.2$ and $K = 0.8$ with the throughput of a standard node. The corresponding curves are given in Fig. 3 (b). This figure shows that when the load is very small (0.1), the throughput of the standard node and the active node when $K = 0.2$ are very similar. But as the load increases, the difference between them increases quickly. The reason is that an increasing load for a standard node means the arrival of more packets which are all passive packets, whereas for an active node, it means an increase in the arrivals of both the passive and, in particular, the active packets since the proportion of passive packets is small and constant. Consider now the case where the proportion of passive packets is much more significant ($K = 0.8$). Clearly the difference between the active and the standard nodes' throughput is significantly reduced.

2. The objective of the second part of the experiments is to show the impact of providing standard nodes with new functionalities on the losses of the standard packets. Fig. 4 illustrates the results obtained.

- In Fig. 4 (a), we see an exponential increase of the loss probability of passive packets as the load increases. This probability increases also as K increases. Note that the probability is less than 0.14 in the worst case. The losses of passive packets are due to the finite capacity of both the input and the output queues. When comparing the loss probabilities in both queues, we have noticed that the exponential increase of the total loss probability is due to the input queue. Indeed, unlike the output queue where the loss probability tends to be constant once a certain load is reached, the loss probability in the input queue is exponential. This difference is due to the fact that a proportion of the packets (active packets) arriving to the input queue remain in the node and are not send to the output queue immediately. Because of the paper length constraint, the figure showing this comparison is not given.



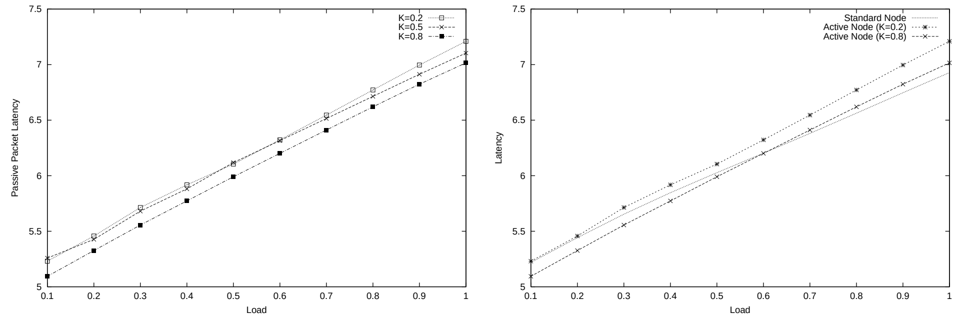
(a) Passive packet loss probabilities *(b) Nodes loss probabilities*

Fig. 4. The loss probabilities

- In Fig. 4 (b), we compare the loss probability of passive packets in an active node and the one obtained for a standard node. We consider both cases for the active node $K = 0.2$ and $K = 0.8$. The figure shows that the loss probabilities for the standard node are bounded by the loss probabilities when $K = 0.2$ and the one when $K = 0.8$. The first one may be considered as the lower bound and the second as the upper bound. This result suggests that the loss of passive packets in an active node depends on the proportion of the active packets arriving at the node. Moreover these losses may be controlled to not exceed a certain limit if we choose the right proportion of active packets to allow in the network.

3. The last performance measure we are interested in is the latency experienced by the passive packets in an active node. The main results we have obtained are summarised in Fig. 5.

- Fig. 5 (a) shows that the proportion $\overline{K}$ of active packets in the node has a negligible effect on the latency experienced by a passive packet. However, as the load increases, this latency increases significantly. A passive packet experiences latency at both the input and the output queue. Active packets arriving at the node are dispatched to the different queues according to their type, but not to the output queue. Therefore, they have an important effect only on the latency experienced by a passive packet in the input queue. But as the latency is computed taking into consideration both queues, the result is that this effect becomes small.
- In Fig. 5 (b), we compare the latency for a passive packet in an active node for $K = 0.2$ and $K = 0.8$ with the latency in a standard node. We can decompose this figure into two parts. The first part considers the case when the load is not heavy (between 0.1 and 6.0) and the second part corresponds to the case where the load is heavy, which is between 0.6 and 1.0.



(a) Passive packet latency *(b) Nodes latency*

Fig. 5. The latency

The first part of the figure suggests that the latency when $K = 0.2$ is an upper bound, and the one when $K = 0.8$ is a lower bound, for the standard node latency. Moreover, the latency in an active node when the proportion of active packets is very high ($K = 0.2$) and the one in a standard node, are similar for a very low load. But, as expected, the latency in the active node becomes higher and the difference between the two curves tends to becomes more significant as the load increases. This is due to the fact that when the load increases, the number of packets arriving at the node increases, in particular, active packets whose proportion is very high when $K = 0.2$. Thus a passive node will have to wait much longer in the input queue. This effect is more striking in the second part of the figure where the load starts to be very high.
When the proportion of active packets is low ($K = 0.8$), the difference, when there is one, between the two latencies is not significant. Note that, in the second part of the figure (high load), the latency in a standard node becomes lower than the one experienced by a passive packet in an active node when $K = 0.8$. The reason is that even if the proportion of active packet is small, there are still active packets arriving at the active node and thus delaying the service of passive packets. This effect is negligible when the load is low (0.1-0.6), and as we have less passive packets ($K = 0.8$) in the active node than in the standard node (100%) the latency in an active node when $K = 0.8$ is a lower bound for the latency in the standard node.

7 Conclusion

In this paper we have used the stochastic process algebra PEPA to investigate an active networking scenario based on active rules. The PEPA model has been used to study, in quantified terms, the impact that the active networking configuration has on the ordinary data packets, at the level of a single node within a network. Such results give us an indication of the disturbance which might be experienced by "*normal*" traffic which must co-exist with "*active*" traffic within a network supporting active networking.

As might be anticipated the presence of active packets, requiring more detailed processing, has a negative impact on the throughput of the node. We assume that packets are lost when either the input or output buffer to the node becomes full, and the second form of impact we have explored is the loss probability. Here we observed that the active packets have less impact on the output buffer than the input buffer, because they are delayed within node awaiting activation. As a result the loss probability for the node becomes dominated by the losses from the input buffer once the load reaches a certain level. Our results appear to suggest that by controlling the ratio of active packets to passive packets it should be possible to bound the loss probability. Finally we studied the latency experienced by passive packets. Here the impact is least marked.

Overall these results suggest that the active networking scenario may be feasible. In our current study the buffer sizes have been relatively small. Increasing the capacity of the different buffers in the node may have an important impact on the rewards, in particular, on the loss probability. Further studies are needed, considering larger buffers and also investigating the impact of varying the arrival rate λ_{gp} of the packets resulting from the execution of the application services. However, we may expect that both parameters do not change the asymptotic behaviour shown in Fig. 4 (b). Nevertheless we have found that PEPA provided a useful modelling formalism within which to carry out this investigation.

Acknowledgements: This work is supported by the CNRS/Royal Society project N.12017.

References

1. Bouzeghoub, M., Kloul, L., Mokhtari, A.: A New Active Network Framework Based on Active Rules. Technical Report, N. 2002/21 PRiSM, Université de Versailles (2002).
2. Calvert, K. (ed.): Architectural Framework for Active Networks. Technical report. AN Architecture Working Group, July 1998.
3. Gilmore, S., Hillston, J.: The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling. In: Proceedings of the Seventh International Conference on Modelling Techniques and Tools for Computer Performance Evaluation. Lecture Notes in Computer Science, Vol. 794. Springer-Verlag, Vienna (1994) 353-368.
4. Hartman, H., Peterson, L., Bavier, A., Bigot, P., Bridges, P., Montz, B., Piltz, R., Proebsting, T., Spatscheck, O.: Joust: A Platform for Liquid Software. In: IEEE Computer Magazine, pp. 50-56, April 1999.
5. Hillston, J.: A Compositional Approach to Performance Modelling. PhD. The University of Edinburgh, 1994.
6. Merugu, S., Bhattacharjee, S., Zegura, E., Calvert, K.: Bowman: A node OS for Active Networks. Proceedings of IEEE Infocom 2000, Tel Aviv, Israel, March 2000.
7. Tullmann, P., Hibler, H., Lepreau, J.: Janos: A Java-oriented OS for Active Networks. IEEE Journal on Selected Areas of Communication, Vol. 19, N. 3, March 2001.