Telecommunication Systems 27:2–4, 413–438, 2004
© 2004 Kluwer Academic Publishers. Manufactured in The Netherlands.

Towards a Feasible Active Networking Scenario

J. HILLSTON and L. KLOUL * {jeh,leila}@inf.ed.ac.uk
LFCS, University of Edinburgh, Kings Buildings, Edinburgh EH9 3JZ, Scotland

A. MOKHTARI amok@prism.uvsq.fr
PRiSM, Université de Versailles, 45, Av. des Etats-Unis, 78035 Versailles Cedex, France

Abstract. We investigate the impact of introducing active networking on traditional packets flowing through the active nodes of a network. Using two approaches, the process algebra formalism PEPA and the simulation package SimJava, 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.

Keywords: active nodes, active rules, performance analysis, packets loss, node latency

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 [Tennenhouse et al., 18] – has shown the necessity of a common architecture able to accept different packet languages and execution environments. The idea of an architecture based principally on three layers has been

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

suggested in [Calvert, 2] and since used in several different works [Hartman et al., 9; Merugu et al., 16; Tullmann et al., 19]. 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 [Bouzeghoub et al., 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 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.

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

To model the nodes and compute the performance measures, we use, on one hand the discrete event, process-oriented simulation package SimJava [McNab and Howell, 15; Howell and McNab, 14], and on the other hand, the process algebra formalism PEPA (Performance Evaluation Process Algebra) introduced in 1994 [Hillston, 10] and widely used since in the performance analysis area [Holton, 13; Gilmore et al., 5; Hillston and Kloul, 11; Fourneau, 3; Gilmore and Kloul, 8].

This paper is structured as follows. In section 2 we present ARFANet, the active-rules based framework for active networks. In section 3, we present the modelling techniques we use for an active-rule based node. In section 4 we present and discuss the results we have obtained. Finally, conclusions and possible extensions of this work are given in section 5.

2. ARFANet framework

ARFANet is an 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.

2.1. 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 or the ACK-Timeout that a node may receive.

  We distinguish between event types (or classes) and event occurrences (or instances). An event type is defined by its name, its attributes and possible constraints on its values. An event instance is defined by an event identifier (evid) and a value (or the values assigned to the list of its attributes).
- *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. If the condition is a logical formula, it is specified within the active rule. 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. If the condition is a boolean function, only its invocation is referred to within the active rule; we assume the definition of the function is made elsewhere using any programming language.
- *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 provision. Examples: send a packet, send an ACK, make a resource allocation, apply a network reconfiguration procedure, etc.

  Actions are considered to be defined elsewhere using any programming language; only their invocation is specified within the active rules.

Rules are organised into modules and an application service is defined by one or several modules. The rules defining an application are to be executed under a certain semantics specified as meta data on the rules. In the following section, we give an overview of the semantics parameters and their values. For more details, see [Bouzeghoub et al., 1].

2.2. Execution semantics specification

Each module of an application is composed of two parts: the header and the effective data. The header of each module provides the set of semantic parameters that govern the behaviour of the active rules constituting the effective data. Table 1 provides a typical structure of an application's module with some of the semantics parameters that may be considered. Here, the value *match* of parameter "Event consumption" specifies that an event occurrence is destroyed only when the condition holds, otherwise it stays indefinitely. The value *immediate* for the E–C coupling mode specifies that the condition part of a rule is immediately evaluated after the event signalling. Similarly, the value *immediate* for the C–A coupling mode specifies that the action part of a rule is immediately

Table 1
Example of module.

Module M1
Execution semantics
Event consumption = match
E–C coupling mode = immediate
C–A coupling mode = immediate
Rule priorities = no
Cascading execution = iterative
Active rules
R$_1$: On E_1 If C_1 Then $A_1()$
R$_2$: On E_1 If C_2 Then $A_2()$
...
R$_n$: On E_p If C_n Then $A_n()$

executed when the condition holds. Moreover, in this example, the parameter "Cascading execution" is defined as *iterative* which means that the cascading execution of the rules is made by a breadth-first approach (at the end of the action, all triggered rules are considered at the same time). For more details about the parameters and their possible values, see [Bouzeghoub et al., 1].

According to these parameters, the execution of an application consists of detecting the events, evaluating the condition and, finally, launching the corresponding actions.

In the following section, we show how to apply the ECA model on the reliable multicast example.

2.3. *Example of the framework instantiation*

A group communication or multicast is a communication which involves more than two participants that want to exchange information. In a multicast system based on active networking, when a packet arrives at a node, if the node has a cache (active), this packet is saved before being transmitted to all the subscribers in the packet group. If a packet is lost during the transmission, the receiver of the packet sends a negative acknowledgment (NACK) which specifies that the packet did not arrive and therefore should be retransmitted.

The arrival of such a packet (NACK) at the active node will trigger either the destruction of the NACK if the missing packet has already been retransmitted, or the retransmission of this packet if it is still in the node cache. In the case that the packet is not in the cache, the node notes the user identity and transmits the first NACK it has received to the source of the lost packet. The packet is either retransmitted by the source itself or by the active node on the link. The packet is deleted from the node if it receives ACK from all clients or the upper Round Time Trip (RTT) is reached.

According to this multicast model, we can define five possible events that may occur in the system (figure 1). These events are described below.

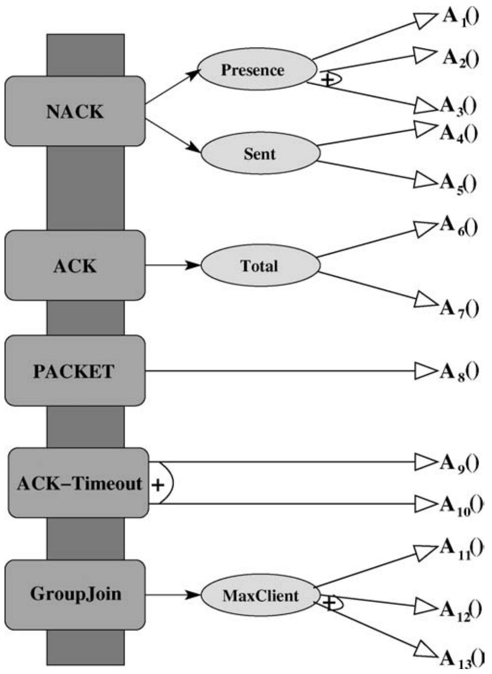


Figure 1. The multicast events list.

1. Event NACK. This event corresponds to the arrival of a negative acknowledgment of a packet from a customer. The processing of such an event will trigger the evaluation of two types of conditions:

- Boolean *presence*. It specifies if the packet is still in the node's cache. If the packet is still in the cache, the node transmits it again to the user who has sent the NACK. This action is denoted $A_1()$. If the packet has been deleted by the node, the identity of the user is noted by the node ($A_2()$) and the first NACK received is then forwarded to the source of the lost packet ($A_3()$). In both cases, the received NACK is consumed at the end of its processing by the node.
- Boolean *sent*. At the arrival of a NACK, the node verifies if the corresponding packet has been already retransmitted to the user. If so, the NACK is not taken into account and is immediately deleted ($A_4()$). If the packet has not been retransmitted yet, the processing corresponding to the Boolean *presence* is triggered ($A_5()$).

2. Event ACK. This event corresponds to the arrival of an acknowledgment of a packet from a customer. The processing to trigger depends on the number of ACK the node has received for the same packet. Let *total*, be this number.

- Integer *total*. If the node has received an ACK from all members of the multicast group concerned with the packet, then the node deletes the packet ($A_6()$), otherwise the node adds the name of the ACK sender to its list of the members who have acknowledged the packet ($A_7()$).

3. Event PACKET. An event of this type has no condition statements to test to trigger the corresponding processing. At the arrival of this event, the node has the list of the subscribers to sent the packet to and sends it according to the routing algorithm used ($A_8()$).

4. Event ACK-Timeout. This event is an example of an internal event that may occur in a system. It is triggered by the operating system of the node. This event has no condition statements to test. The node supposes that the customer is not able to receive other packets for different reasons such as the corruption or the congestion of the link, or simply because the customer is disconnected. Therefore, the node has to remove the customer from the list of the subscribers ($A_9()$) and to forward this information to the server ($A_{10}()$).

5. Event GroupJoin. This event corresponds to the arrival, from a new customer, of a request to join the group. The processing to trigger depends on the maximal number of subscribers that can be accepted by the server. Let *MaxClient* be this number.

- Integer *MaxClient*. If the maximal number of multicast group members is reached, then the node rejects the new customer request ($A_{11}()$), otherwise the node adds the new customer to its list ($A_{12}()$) and sends this information to the server ($A_{13}()$).

The active rules describing the multicast model and their execution semantics are given in table 2. In this example, rules R_1, R_2, R_3 and R_4 have the same priority.

Note that the strategy of any application can be easily modified. Thus, we can, for example, add rules or disable some of the existing ones. We can also change the execution semantics by, for example, changing the priority of the rules or the value of a parameter like replacing the value *match* of the parameter event consumption in the multicast example by the value *always*. In this case, instead of destroying each event occurrence only when the condition holds, it is destroyed after its consideration whether the condition holds or not. In [Bouzeghoub et al., 1], several application examples are developed, showing the flexibility offered by the approach to define and modify an application strategy.

2.4. Active node architecture

Each node of the network is defined by a layered architecture [Bouzeghoub et al., 1] as shown in figure 2. Besides the operating system, it is composed of three layers: the *application services*, the *active monitors* and the *virtual machine*.

- *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

Table 2
Multicast module.

Module M1: Multicast
Execution semantics
Event consumption = match
E–C coupling mode = immediate
C–A coupling mode = immediate
Rule priorities = (R_3, R_4, R_1, R_2); R_5; R_6; R_7; R_8; R_9; R_{10}
Cascading execution = iterative
Active rules
R_1: On NACK If *Presence* Then $A_1()$
R_2: On NACK If $\neg$*Presence* Then $\{A_2(), A_3()\}$
R_3: On NACK If *Sent* Then $A_4()$
R_4: On NACK If $\neg$*Sent* Then $A_5()$
R_5: On ACK If $total = Max$ Then $A_6()$
R_6: On ACK If $total < Max$ Then $A_7()$
R_7: On PACKET If *True* Then $A_8()$
R_8: On ACK-Timeout If *True* Then $\{A_9(), A_{10}()\}$
R_9: On GroupJoin If $number = MaxClient$ Then $A_{11}()$
R_{10}: On GroupJoin If $number < MaxClient$ Then $\{A_{12}(), A_{13}()\}$

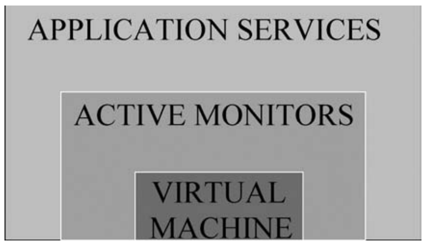


Figure 2. Active node architecture.

requiring the same execution semantics. Active monitors are dynamically loaded in the node.

- *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 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 cor-

responding 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. The header of each application module is used during the binding process to allocate an appropriate active monitor to the given module.
– binding application services to active monitors in active nodes or application data to application services.

The virtual machine layer does not exist in the architecture introduced in [Calvert, 2], at least not explicitly. The explicit definition of this layer in ARFANet allows us to add a certain flexibility to the node configuration. It allows the definition of a network architecture where active nodes may be deployed dynamically, for example, according to the congestion state of the network. Indeed, in ARFANet the virtual machine is a generic component which has the same functionality in every active node but an implementation which depends on each active node system environment. The virtual machine is implemented consistently in each node of the network. A node provided with a virtual machine is not automatically an active node. It may be considered as a standard node as long as it does not receive a service packet such as an active monitor packet, making it active.

3. Modelling the active node

Modelling the active node architecture of figure 2 and its execution mode assumes that the node is viewed as a set of finite queues as depicted in the configuration of figure 3. The packets arriving at the node are queued in $queue_{\text{in}}$, before being dispatched according to their type as follows:

- Passive packets (to forward) are directly shortcut to the outgoing queue, called $queue_{\text{out}}$. We denote by q_p the probability that a packet arriving at the input queue is a standard packet.
- Active data packets (*d-packets*) are oriented to the events queue denoted $queue_{\text{event}}$. We denoted by q the probability that a *d-packet* references an active code which is not present in the node.
- Application service packets (*s-packets*) are oriented to the service queue denoted $queue_s$.
- Active monitor packets (*m-packets*) are oriented to the monitors queue denoted $queue_m$.

Besides the *d-packets*, $queue_{\text{event}}$ receives also internal events such as the events generated by the system and the application services. If a *d-packet* in service in $queue_{\text{event}}$ references a non-existent active code packet (*s-packet* or *m-packet*) in the node, the packet is requeued in $queue_{\text{event}}$ and a request for the missing packet(s) is sent to the

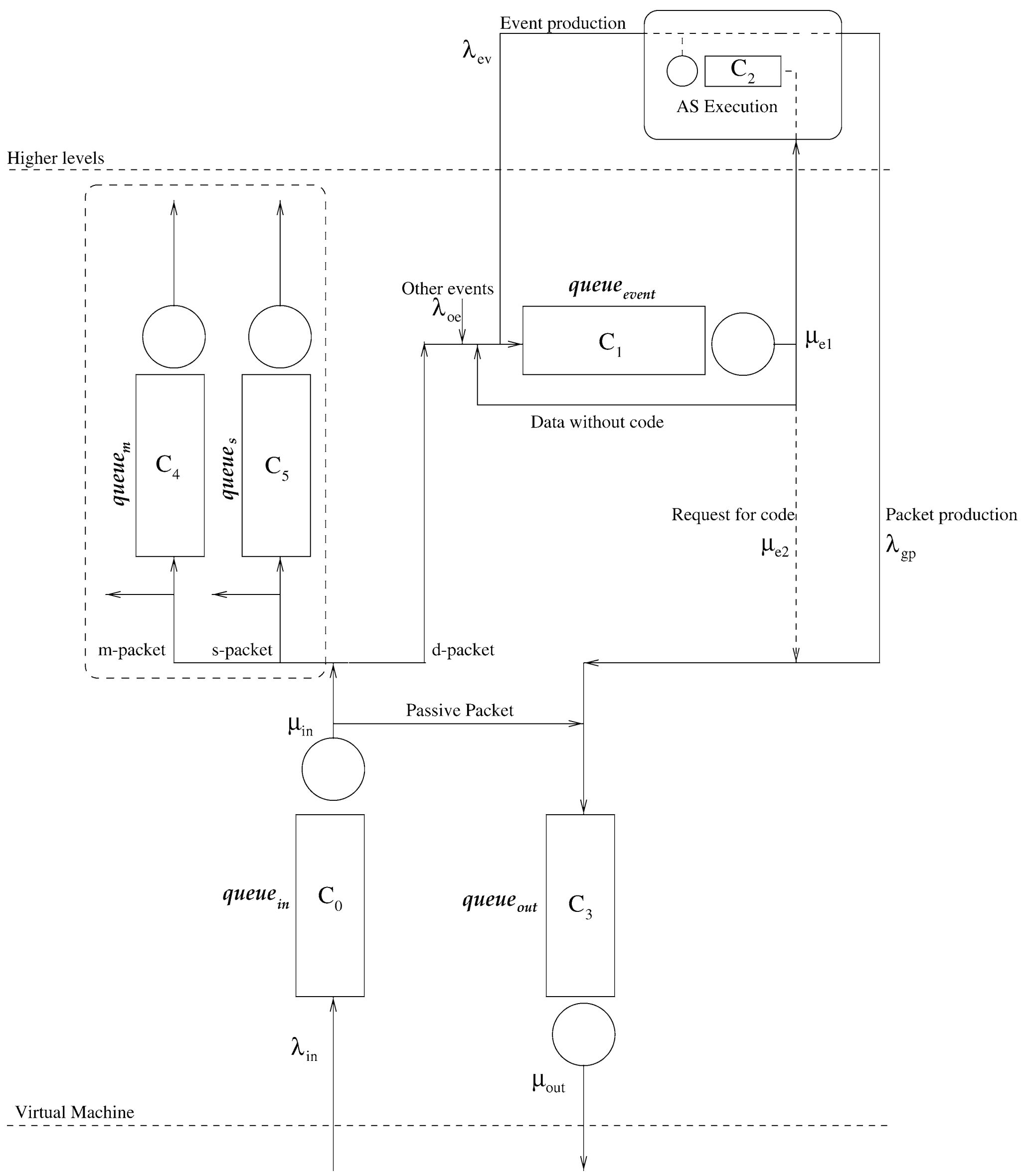


Figure 3. Queueing network of the active node.

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.

The objective of this study is to analyse the impact of the active networking on the standard or passive packets performance. As these packets are immediatly shortcut from the input to the output queue, only the architecture layers having a direct link, thus an impact, with respect to these queues are explicitly represented. In this configuration (figure 3), besides the *virtual machine*, the *application service* layer is represented by *queue*$_{as}$ which receives from the virtual machine the necessary data for an application execution, and which may generate packets towards the output queue. The layer *active monitors* is not explicitly represented because it has no direct link with the outgoing queue of the node.

We consider two approaches to model and evaluate the performance of this queueing network. The first approach consists of using the process algebra formalism PEPA (Performance Evaluation Process Algebra) whereas the second approach consists of using SimJava simulation. Both approaches are briefly described in the following.

3.1. *Analytical modelling using PEPA*

The first approach consists of using the process algebra formalism PEPA (Performance Evaluation Process Algebra). PEPA models are described as interactions of *components*. Each component can perform a set of actions. A random variable, representing duration, is associated 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. The PEPA model is developed in [Hillston et al., 12] and is solved using the PEPA Workbench [Gilmore and Hillston, 4].

3.2. *Simulation using SimJava*

The second approach consists of using SimJava simulation [McNab and Howell, 15; Howell and McNab, 14]. SimJava is a discrete event process-oriented simulation package developed from earlier C++ simulation tools. It aims to make simulation models more widely available and to allow operation across the Internet using Java applets. A SimJava simulation consists of a number of entities each of which run in parallel in their own thread. The entities are connected together by ports and communicate by sending and receiving event objects through these ports. A static class controls the threads and advances the simulation time. SimJava is a collection of three packages:

- a package which provides the basic discrete event simulation,
- a package which provides a framework for displaying simulation as an applet,
- and finally a package which provides modules for displaying statistical results.

3.3. *Performance criteria*

The performance measures of the system we are interested in are the throughput, the loss rate 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. 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 using on one hand the PEPA Workbench and on the other the SimJava package.

4. Numerical results

To compute the 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 $\lambda_{\text{in}} = \lambda_{\text{in,a}} + \lambda_{\text{in,s}}$, where $\lambda_{\text{in,a}}$ is the arrival rate of active packets and $\lambda_{\text{in,s}}$ the arrival rate of standard packets. As we are interested in the performance of the node with respect to the standard packets, we define $\lambda_{\text{in,s}} = K \times \lambda_{\text{in}}$ where K is the proportion of standard packets which varies between 0.2 and 0.8. Regardless of the value of K, the active packets are divided between the different types of active packets according to the proportions defined in table 3. The active code represented by the *m-packets* and the *s-packets*, is involved only in the active rules execution, not in their managing or their launching. The persistence of the active code in the cache and the modularity of the active rules allows for intensive reuse, resulting in a small traffic of code in the network compared to the active data. This explains the proportions defined in table 3.

Assuming that the average speed of a switch in the network is 10 Mbits/s and the average packet size is 250 bytes, we obtain an average service rate in the queues of the node of 5000 packets/s, except for $queue_{\text{as}}$ the queue of the applications layer whose server is assumed to be faster, with a rate equal to 7000 packets/s.

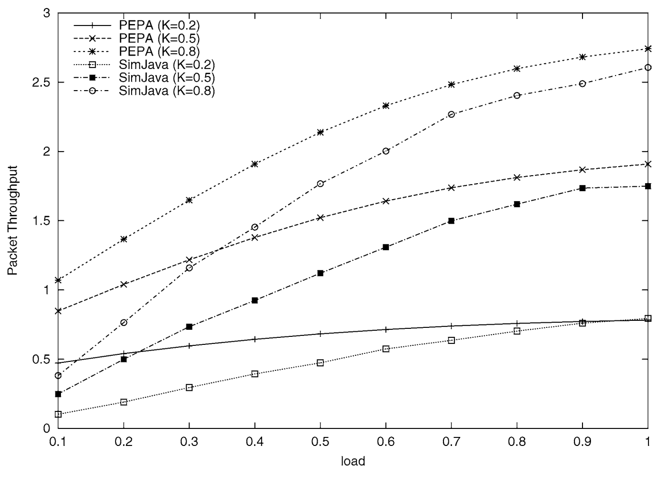
In our initial experiments, we assume that all buffers have the same capacity $C_i = 5$, $i = 0, \ldots, 3$. This is the minimal size considered in this study; to analyse the impact of the buffer size on the delays and the packet loss, other buffer sizes are considered in section 4.1. To minimize the delays, it is recommended to consider the smallest sizes possible, up to few dozens of packets [Morris, 17] depending on the network throughput. Our experiments take place in this context.

As all buffers have the same capacity, in the following, we denote by C the size of these buffers. For a buffer capacity $C = 5$, the resulting Markov chain has 26,136 states and 214,632 transitions. The parameters of the model are summarised in table 3.

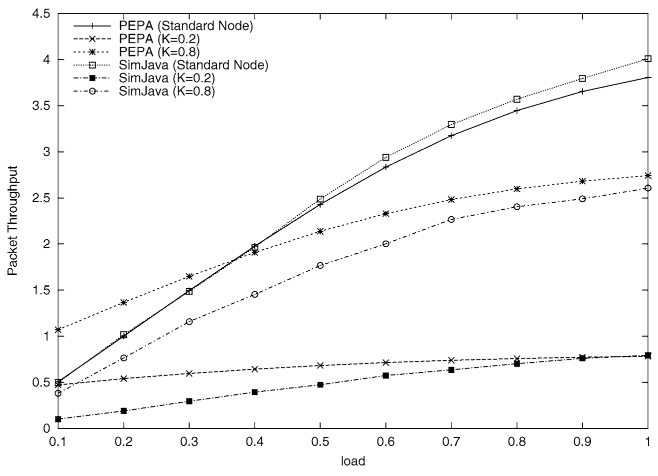
The results obtained are depicted in figures 4–13. All curves are plotted versus the load of the input queue; we decrease the inter-arrival delay of packets ($1/\lambda_{\text{in}}$), thus

Table 3
Input parameters.

Active packets repartition	Arrival rates (p/ms)	Service rates (p/ms)	Probabilities
d-packet: 75%	$\lambda_{\text{ev}} = 1.0$	$\mu_{\text{in}} = 5.0$	$q_p = K$
m-packet: 12.5%	$\lambda_{\text{oe}} = 1.0$	$\mu_{\text{out}} = 5.0$	$q = 0.5$
s-packet: 12.5%	$\lambda_{\text{gp}} = 1.0$	$\mu_{\text{e}} = 5.0$; $\mu_{\text{as}} = 7.0$	



(a)



(b)

Figure 4. The throughput for the standard packets. (a) Impact of K on the throughput; (b) standard node versus active node.

increasing the load on the node. The confidence interval of each simulation run was 95%. We do not show the intervals in the graphs since we are interested in the trends.

1. The throughput. In the first part of our experiments, we are interested in the throughput of the active node for the passive packets.

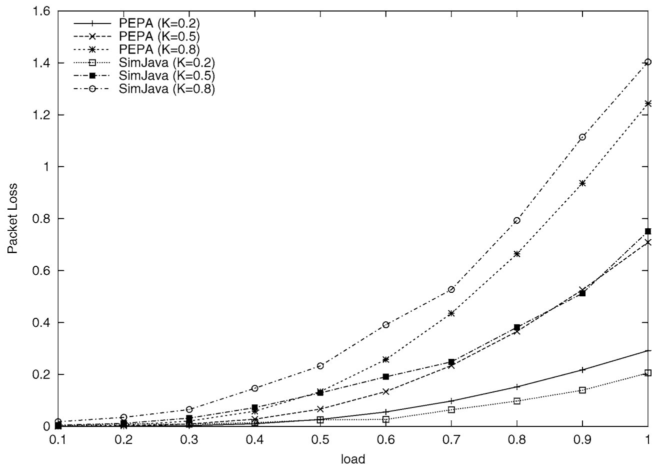
- Figure 4(a) depicts the impact of K on the throughput of the node for standard packets. Obviously this throughput increases as the proportion of passive packets K increases and also as the arrival rate to the input queue λ_{in} increases. Note that when the input queue becomes heavily loaded the throughput tends to be stable, for all values of K. These results are valid for both simulation and analytical modelling (PEPA).

- To have an idea of the impact on passive packets' throughput of providing standard nodes with new functionalities, 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. Figure 4(b) shows that when the load is very small, the throughput of the standard node and the active node when $K = 0.2$ are quite 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.

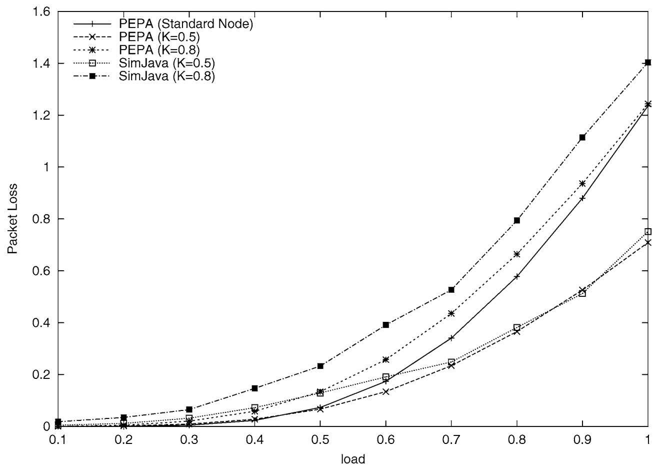
In the rest of the paper, we will consider only the PEPA results for the standard node because the SimJava and the PEPA results are in all cases almost the same, as is shown in figure 4(b).

2. The loss rates. The objective of the second part of the experiments is to show the impact of providing standard nodes with new functionalities on the loss of standard packets. The results obtained are shown in figure 5.

- In figure 5(a), we see an exponential increase in the loss rate of passive packets as the load increases and also as K increases. The losses of passive packets are due to the finite capacity of both the input and the output queues. The exponential increase of the total loss rate is due to the input queue. Indeed, unlike the output queue, where the loss rate tends to be constant once a certain load is reached, the loss rate in the input queue is exponential. This difference is due to the fact that a proportion of the packets (active packets) arriving at the input queue remain in the node and are not sent to the output queue immediately. Note that the simulation and the analytical model results are very similar.
- In figure 5(b), we compare the loss rate of passive packets in an active node and the one obtained for a standard node. For the active node, we consider both cases $K = 0.5$ and $K = 0.8$. The figure shows that the loss probabilities for the standard node are bounded by the loss probabilities when $K = 0.5$ 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 is noticeable for both simulation and PEPA results. 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.



(a)

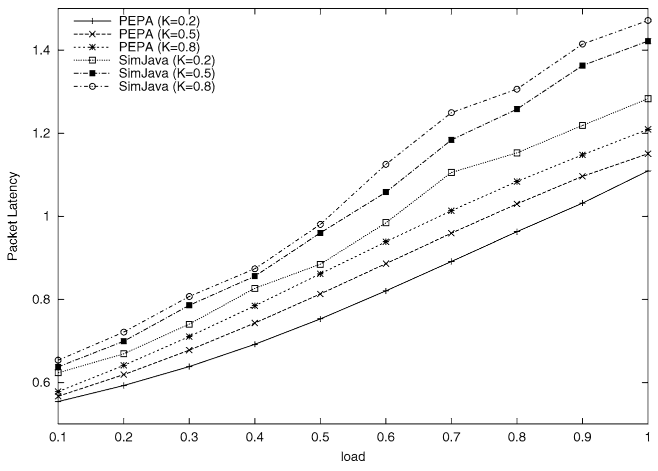


(b)

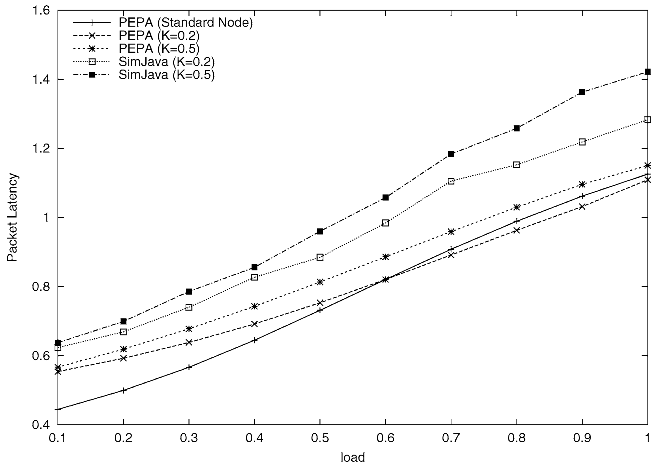
Figure 5. The loss rate of standard packets. (a) Impact of K on the loss rate; (b) standard node versus active node.

3. The latency. The last performance measure we are interested in is the latency experienced by the passive packets in an active node. Figures 6(a) and (b) show the results we have obtained.

- 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 effect only on the latency experienced by a passive packet in the input queue. As the latency is computed taking into consideration both queues, the increase of the proportion K may be an additional effect. Indeed increasing K implies increasing the number of passive packets in the



(a)



(b)

Figure 6. The latency for the standard packets. (a) Impact of K on the latency; (b) standard node versus active node.

output queue and thus the latency in this queue. Figure 6(a) shows this phenomenon as the latency experienced by a passive packet increases as the proportion of active packets $\overline{K}$ decreases. However, it shows also that the effect of $\overline{K}$ on the latency is negligible. This conclusion is more striking for the PEPA results than for the simulation.

- In figure 6(b), we compare the latency for a passive packet in an active node for $K = 0.2$ and $K = 0.5$ with the latency in a standard node. This figure shows overall that the difference between these latencies is negligible, when considering the PEPA results. We can decompose this figure into two parts. The first part considers the case when the load is not heavy (between 0.1 and 0.6) and the second part corresponds to the case where the load is heavy, which is between 0.6 and 1.0. In the first part of the figure (low load), the latency in a standard node is lower than the one experienced by a passive packet in an active node and this for all values of K. The reason is that even if the proportion of active packets is small, there are still active packets arriving at the active node and thus delaying the service of passive packets. This effect becomes negligible when the load is high in both nodes. The second part of the figure suggests that the latency when $K = 0.2$ is a lower bound, and the one when $K = 0.5$ is an upper bound, for the standard node latency.

4.1. Impact of the buffers capacity

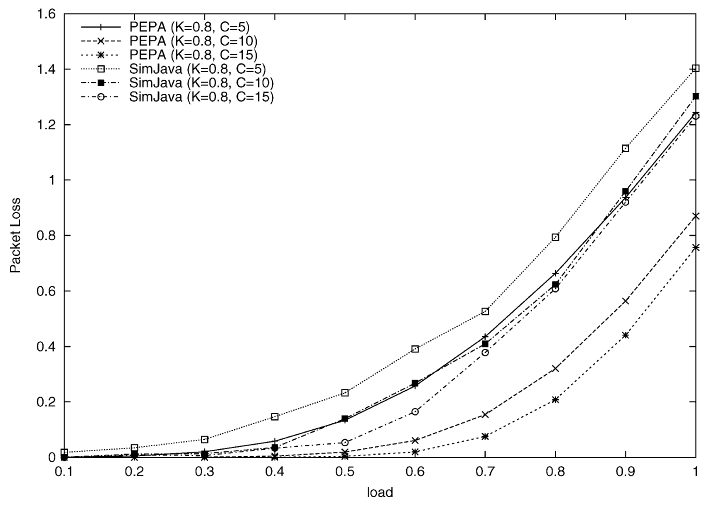
In these experiments, we investigate the impact of increasing the buffer size on both the passive packets' loss and the latency experienced by these packets. For that we consider two cases, $C = 10$ and $C = 15$. The results obtained are depicted in figures 7, 8.

1. The loss rates. Figure 7 shows the impact of the buffer capacity on the passive packets' loss rate.

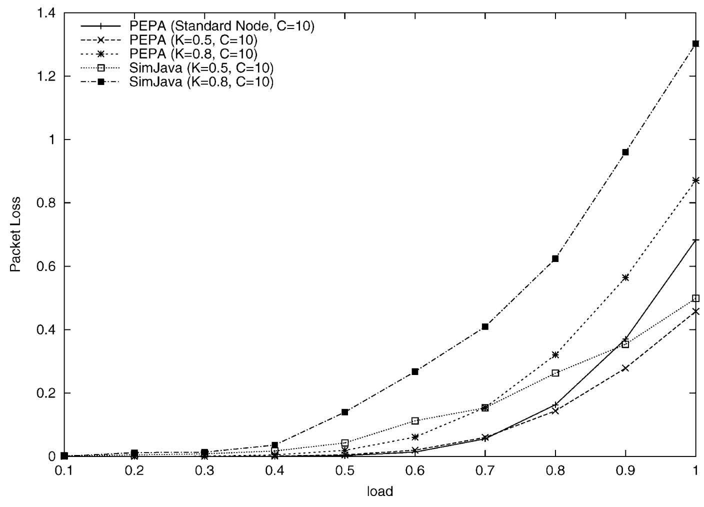
- In figure 7(a), we compare the loss rate of passive packets for the three buffer sizes $C = 5, 10, 15$. The measures are made for the proportion of passive packets in the node $K = 0.2$, that is when the traffic of standard packets is very perturbated. This figure shows first that the loss rate increases exponentially as the load increases for all buffer capacities. Moreover this rate decreases as the buffer capacity increases.

  However, we can notice that the difference between the loss rate when $C = 10$ and $C = 15$ is negligible for a load between 0.1 and 0.6. This difference starts to increase as the load becomes greater than 0.6, but it always remains much smaller than the difference between the losses when $C = 10$ and $C = 5$. This suggests that if we continue to increase the buffers' size, the loss rate will perhaps continue to decrease. However, it seems likely that we will reach a buffer capacity at which this rate will no longer significantly decrease.
- In figure 7(b) we compare the loss rate in a standard node with the one in an active node. This comparison is done for a buffer capacity $C = 10$ and different values of K: 0.5, 0.8. Figure 7(b) confirms the results we have obtained in figure 7(a) for $C = 5$, as the loss rate in a standard node can be bounded by the loss rate in an active node. Once again this suggests that the loss of passive packets in an active node may be controlled to not exceed a certain limit if we choose the right proportion of active packets to allow in the network. We can see that $K = 0.8$ may be used as an upper bound as in figure 7(a), and it is better to use $K = 0.5$ rather than $K = 0.2$ as a lower bound. Of course these bounds may still be refined in both cases $C = 5$ and $C = 10$.

2. The latency. In the following, we show using figure 8 the impact of the buffer capacity on the latency experienced by a standard packet in an active node.



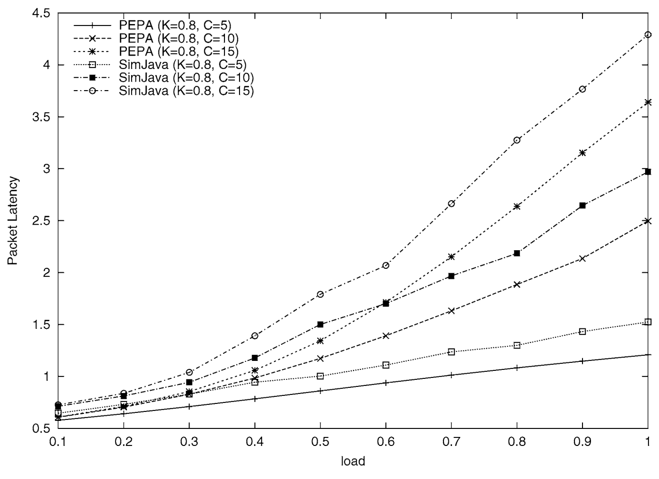
(a)



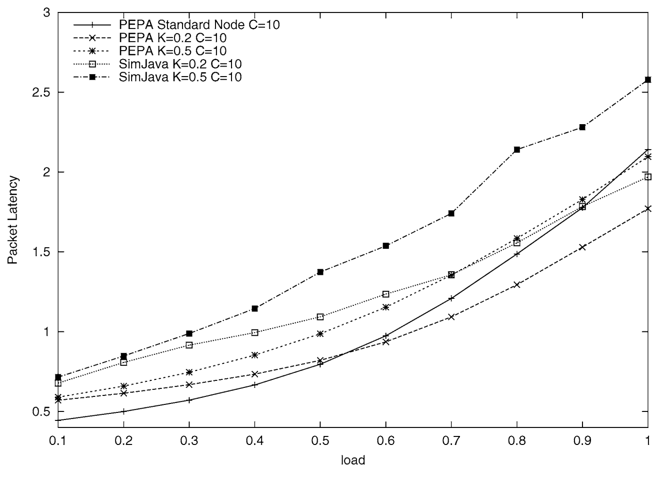
(b)

Figure 7. The loss rate for standard packets. (a) Impact of C when $K = 0.8$; (b) standard node versus active node.

- In figure 8(a), we compare the latency obtained for the different buffer sizes $C = 5$, 10 and 15. The proportion of active packets is $\overline{K} = 0.8$, that is the standard packet traffic is very perturbed. This figure shows that the latency increases as the load increases and this for all values of buffer capacity. We can notice that when the load is low (0.1–0.5), the impact of buffer capacity is not significant, but as the load increases, this impact is much more important. The latency increases proportionally to the buffer size.



(a)



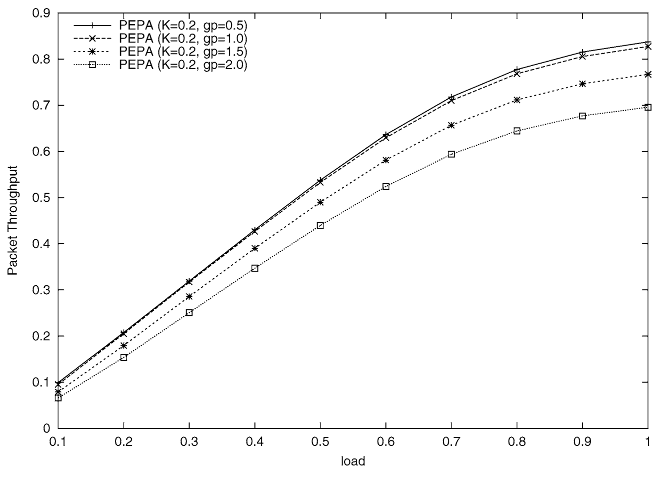
(b)

Figure 8. The latency for the standard packets. (a) Impact of C when $K = 0.8$; (b) standard node versus active node.

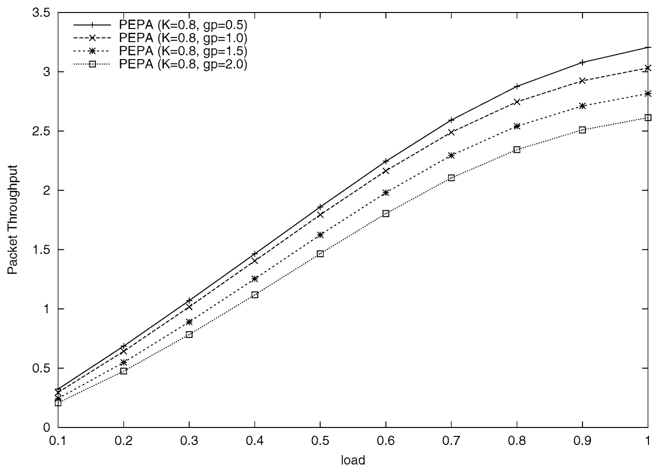
- In figure 8(b), we compare the latency experienced by a standard packet in both types of nodes. We consider buffers' capacity $C = 10$ and two values of K: 0.2 and 0.8. Figure 8(b) depicts the same behaviours as those seen in figure 6(b).

4.2. Impact of the active packets generated in the node

To investigate the impact of the active packets resulting from the execution of active rules in the application service layer, we consider different values of the arrival rate λ_{gp} to the output queue: 0.5, 1.0, 1.5, and 2. The results obtained for buffer capacity $C = 10$



(a)

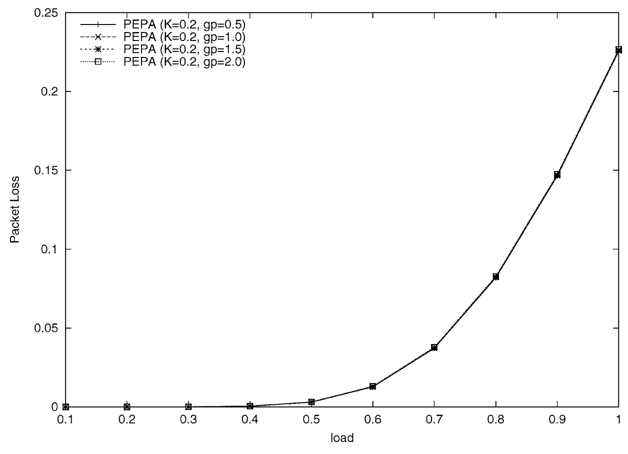


(b)

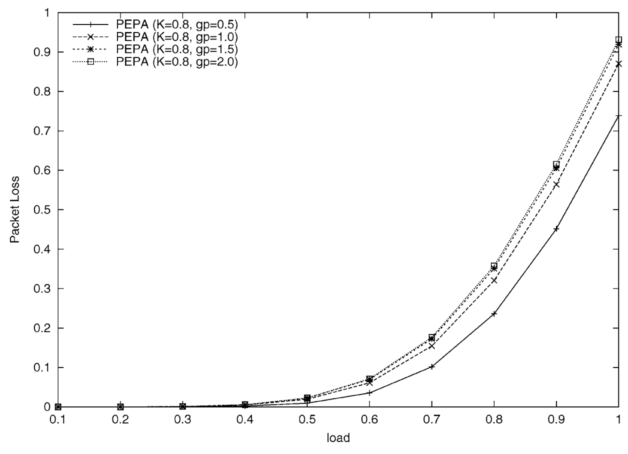
Figure 9. The standard packets throughput. (a) Scenario 1: $K = 0.2$; (b) scenario 2: $K = 0.8$.

are depicted in figures 9–12. As there is a good agreement between the simulation and the PEPA model results, we only show the results of the PEPA model.

1. The thoughput. Figures 9(a) and (b) show the throughput obtained for $K = 0.2$ and $K = 0.8$, respectively. In both scenarios, the throughput decreases as λ_{gp} increases. Moreover this decrease becomes more significant as this rate increases, in particular when the node is loaded and the standard packets traffic is very perturbed ($K = 0.2$). The more active packets we have in the node, the more likely the generation of active packets in this node is.



(a)

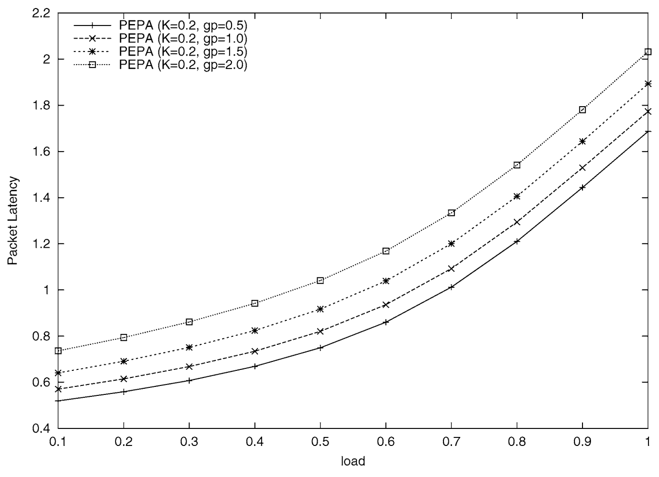


(b)

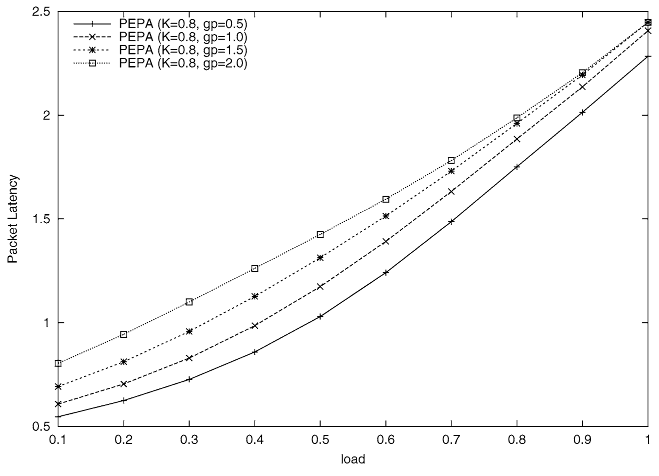
Figure 10. The standard packets loss rate. (a) scenario 1: $K = 0.2$; (b) scenario 2: $K = 0.8$.

2. The loss rate. Figure 10(a) shows the loss rate of passive packets when $K = 0.2$ and for all values of λ_{gp}. In this figure all curves are superposed for all values of the load and λ_{gp}. This result implies that the generation of active packets inside the node has no effect on the losses when the proportion of passive packet is very small. The same phenomenon is observed when $K = 0.8$ (figure 10(b)) but only when the load is small (below 0.5). However, figure 10(b) shows that as λ_{gp} increases, the effect on the losses quickly becomes insignificant.

3. The latency. As for both previous performance measures, we analyse the node latency for passive packets for all values of λ_{gp}. Figure 11(a) shows the latency when the proportion of passive packets is $K = 0.2$. In this figure, all curves have the same behaviour but as λ_{gp} increases, the latency increases too. The increase of the latency



(a)



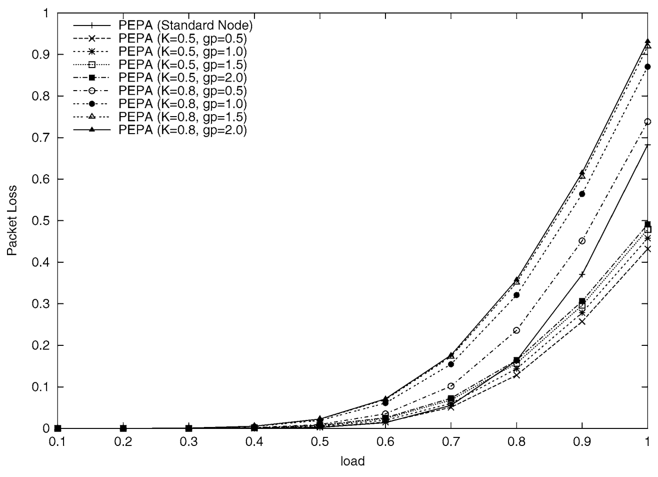
(b)

Figure 11. The standard packets latency. (a) Scenario 1: $K = 0.2$; (b) scenario 2: $K = 0.8$.

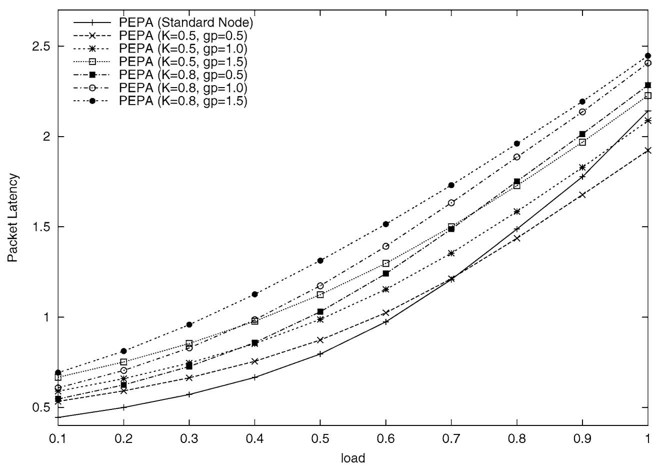
becomes more significant as $\lambda_{\rm gp}$ increases. When $K = 0.8$, the same phenomenon can be observed in figure 11(b). However this one shows that as the node becomes loaded the increase of the latency becomes insignificant as $\lambda_{\rm gp}$ increases.

4. Standard node versus active node. In order to compare the loss rate and the latency of passive packets in an ARFANet node and in a standard one, we vary the active packets generation rate for both proportions $K = 0.5$ and $K = 0.8$.

- In figure 12(a) we can see that as the variation of $\lambda_{\rm gp}$ has almost no impact on the losses when $K = 0.5$, this one can be considered as a good lower bound, in particular for $\lambda_{\rm gp} = 0.5$. This confirms the previous results we have obtained (section 4).



(a)



(b)

Figure 12. Standard node versus active node. (a) Loss rate of passive packets; (b) latency for passive packets.

Similarly this figure shows that $K = 0.8$ can be considered as a good upper bound for the losses, but for $\lambda_{\mathrm{gp}} = 0.5$. If a bigger value of λ_{gp} is considered, we may have to choose a bit smaller value of K than 0.8.

- Figure 12(b) shows the latency for the three first values of λ_{gp} (0.5, 1.0, 1.5) and the standard packets proportions $K = 0.5$ and $K = 0.8$. This latency is compared with the one in a standard node. We can see that for both values of K, the best results are obtained when $\lambda_{\mathrm{gp}} = 0.5$. Another value of λ_{gp} may be used, like 1.0, but we may have to reduce the value of K.

These experiments allow us to show that the generation of active packets in the node has an impact on the performance of standard packets. However, this impact is not significant for the losses because most of them occur in the input queue whereas the generated active packets are queued in the output queue. The generation of new active packets by the node influences significantly the latency of passive packets because the presence of more packets in the output queue delays the service of the standard packets present.

We can conclude that the active packets' generation rate and the proportion of these packets arriving at the node are closely related and the choice of one has an impact on the choice of the other. Moreover both must be limited to a certain value in order to achieve performance comparable to a standard node.

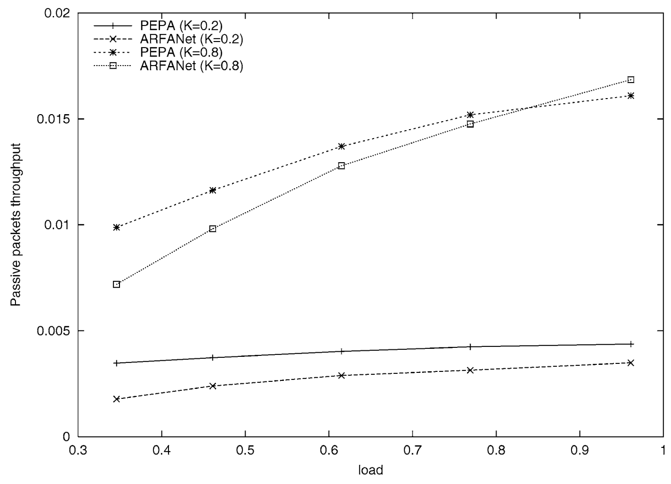
4.3. Comparing the analytical results with the real measures

In this section, we compare the results obtained using the PEPA model with the measures made on the ARFANet framework. To make this comparison we had to measure the real service time of the infrastructure and adapt the parameters of the PEPA model to this new service rate (26 packets/s). Our tests have been made on a Linux machine with an "AMD Athlon(tm) XP 1500+" processor of 1339 MHz, 256 K of memory cache and 256 M memory.

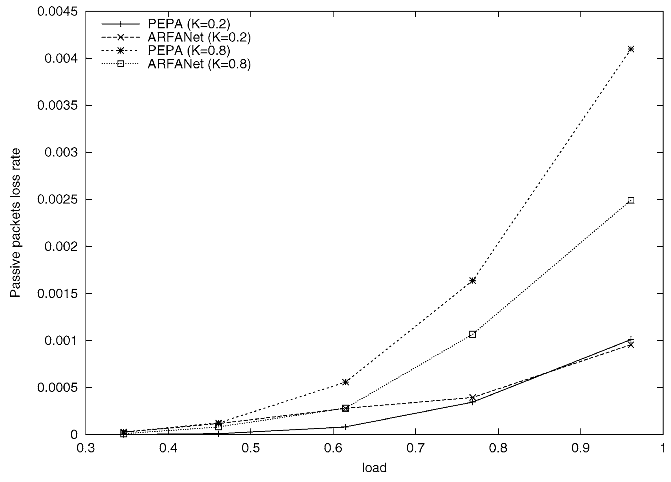
Figure 13 shows the results obtained for the throughput and the loss rate when $C = 10$ and considering both proportions of standard packets $K = 0.2$ and $K = 0.8$.

- In figure 13(a), we compare the throughput obtained from the PEPA model and that measured on ARFANet. This figure shows that these results are very similar for both values of K. When $K = 0.2$, the throughput of the PEPA model and of ARFANet have the same behaviour, the former being always greater than the latter. However when the traffic of standard packets is slightly perturbed ($K = 0.8$), the PEPA model throughput is greater only in the first half of the load. In the second half, the phenomenon is the opposite.
- For the losses, we see in figure 13(b) that when $K = 0.2$ the PEPA model losses are very similar to the implementation and this for all values of the load. However, when $K = 0.8$, both loss rates are initially very close ($load < 0.7$). As the load increases, the distance between the two curves increases, the loss rate of the PEPA model being always a bit greater. Note that for the sake of readability of the first part of the figure (0.3–0.7), the maximal load is limited to 1.0.

If the analytical analysis allows us to have a control on the parameters choice, in particular, on the service rate, in the case of the implementation, this one is imposed by the hardware configuration of the machine used which does not always correspond to the reality. However, globally the results of the tests we have made confirm those obtained using the analytical technique PEPA, and thus confirm the analytical analysis of the impact of introducing active packets on the standard packets.



(a)



(b)

Figure 13. PEPA results versus ARFANet measures. (a) The throughput for passive packets; (b) the loss rate of passive packets.

5. Conclusion

In this paper we have used both the stochastic process algebra PEPA and the object-oriented simulation technique SimJava to investigate an active networking scenario based on active rules called ARFANet. These techniques have 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.

We explore different forms of impacts: the throughput, the loss rate and the latency experienced by the standard packets in the node. Different buffer sizes have been considered. For the loss rate, our results appear to suggest that by controlling the ratio of active packets to passive packets it should be possible to bound the loss rate.

Similarly the results obtained for the latency suggest that it should be possible to bound it by controlling the proportion of active packets in the node. Moreover as the buffers' capacity increases the impact on the latency becomes more significant.

Overall these results suggest that the active networking scenario is feasible. The implementation of the ARFANet framework is in its final stage. Further work consists of testing the plateform's performance and comparing the results obtained with those presented in this paper.

An extension of this work will be to consider a network of nodes, in order to study the end-to-end impact on standard packets of introducing active nodes and active packets in the network. We believe an appropriate formalism for this will be PEPA nets [Gilmore et al., 6, 7], a combination of PEPA and Stochastic Petri Nets.

Acknowledgement

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

References

[1] M. Bouzeghoub, L. Kloul and A. Mokhtari, A new active network framework based on active rules, Technical Report, N. 2002/21 PRiSM, Université de Versailles (2002).

[2] K. Calvert, ed., Architectural framework for active networks, Technical Report, AN Architecture Working Group (July 1998).

[3] J.-M. Fourneau, L. Kloul and F. Valois, Performance modelling of hierarchical cellular networks using PEPA, Performance Evaluation 50(2/3) (2002) 83–99.

[4] S. Gilmore and J. Hillston, The PEPA workbench: A tool to support a process algebra-based approach to performance modelling, in: *Proc. of the 7th Internat. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, Lecture Notes in Computer Science, Vol. 794 (Springer, Vienna, 1994) pp. 353–368.

[5] S. Gilmore, J. Hillston, D.R.W. Holton and M. Rettelbach, Specifications in stochastic process algebra for a robot control problem, International Journal of Production Research 34(4) (1996) 1065–1080.

[6] S. Gilmore, J. Hillston and L. Kloul, PEPA nets, in: *Performance Tools and Applications to Networked Systems: Revised Tutorial Lectures*, eds. M.C. Calzarossa and E. Gelenbe, Lecture Notes in Computer Science, Vol. 2965 (Springer, New York, 2004) pp. 311–335.

[7] S. Gilmore, J. Hillston, L. Kloul and M. Ribaudo, PEPA nets: A structured performance modelling formalism, Performance Evaluation 54 (2003) 79–104.

[8] S. Gilmore and L. Kloul, A unified tool for performance modelling and prediction, in: *22nd Internat. Conf. on Computer Safety, Reliability and Security*, Lecture Notes in Computer Science, Vol. 2788 (Springer, Edinburgh, 2003) pp. 179–192.

[9] H. Hartman, L. Peterson, A. Bavier, P. Bigot, P. Bridges, B. Montz, R. Piltz, T. Proebsting and O. Spatscheck, Joust: A platform for liquid software, IEEE Computer Magazine (April 1999) 50–56.

[10] J. Hillston, A compositional approach to performance modelling, Ph.D. thesis, The University of Edinburgh (1994).

[11] J. Hillston and L. Kloul, Performance investigation of an on-line auction system, Concurrency and Computation: Practice and Experience 13(1) (2001) 23–41.

[12] J. Hillston, L. Kloul and A. Mokhtari, Active nodes performance analysis using PEPA, in: *Proc. of the 9th UK Performance Engineering Workshop (UKPEW'03)*, Warwick, UK (July 2003).

[13] D.R.W. Holton, A PEPA specification of an industrial production cell, in: *Proc. of the 3rd Internat. Workshop on Process Algebras and Performance Modelling*, Special Issue of Computer Journal 38(7) (1995).

[14] F. Howell and R. McNab, SimJava: A discrete event simulation package for Java with applications in computer systems modelling, in: *Proc. of the 1st Internat. Conf. on Web-based Modelling and Simulation*, San Diego, CA, Society for Computer Simulation (January 1998); SimJava URL: `http://www.dcs.ed.ac.uk/home/hase/simjava`.

[15] R. McNab and F.W. Howell, Using Java for discrete event simulation, in: *Proc. of the 12th UK Computer and Telecommunications Performance Engineering Workshop (UKPEW)*, Edinburgh (1996).

[16] S. Merugu, S. Bhattacharjee, E. Zegura and K. Calvert, Bowman: A node OS for active networks, in: *Proc. of IEEE INFOCOM 2000*, Tel Aviv, Israel (March 2000).

[17] R.T. Morris, Scalable TCP congestion control, Ph.D. thesis, Harvard University, Cambridge, MA (1999).

[18] D. Tennenhouse, J. Smith, D. Sincoskie, D. Wetherall and G. Minden, A survey of active network research, IEEE Communications Magazine 35(1) (1997) 80–86.

[19] P. Tullmann, H. Hibler and J. Lepreau, Janos: A Java-oriented OS for active networks, IEEE Journal on Selected Areas in Communications 19(3) (2001).