

Towards a Feasible Active Networking Scenario

J. Hillston¹, L. Kloul^{1*} and A. Mokhtari²

¹LFCS

²PRiSM

University of Edinburgh

Université de Versailles

Kings Buildings

45, Av. des Etats-Unis

Edinburgh EH9 3JZ, Scotland

78035 Versailles Cedex, France

{jeh,leila}@inf.ed.ac.uk

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 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 [1]—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 suggested in [2] and since used in several different works [3][4][5]. 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 [6]. 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-

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

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 [7] [8], and on the other hand, the process algebra formalism PEPA (Performance Evaluation Process Algebra) introduced in 1994 [9] and widely used since in the performance analysis area [10][11][12][13][14].

This paper is structured as follows. In Section 2 we present 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.

- *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 provision. Examples: send a packet, send an ACK, 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 [6] for more detail). According to these parameters, the execution of an application consists of detecting the events, evaluating the condition and finally launching the corresponding actions.

2.2 Active Node Architecture

Each node of the network is defined by a layered architecture [6] 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*.

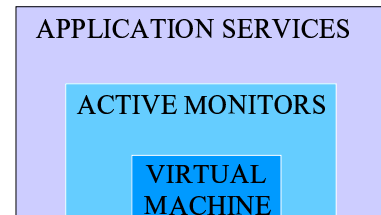


Figure 1: Active Node Architecture

- *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 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 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.

The virtual machine layer does not exist in the architecture introduced in [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 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:

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

Besides the *d-packets*, $queue_{event}$ receives also internal events such as the events generated by the

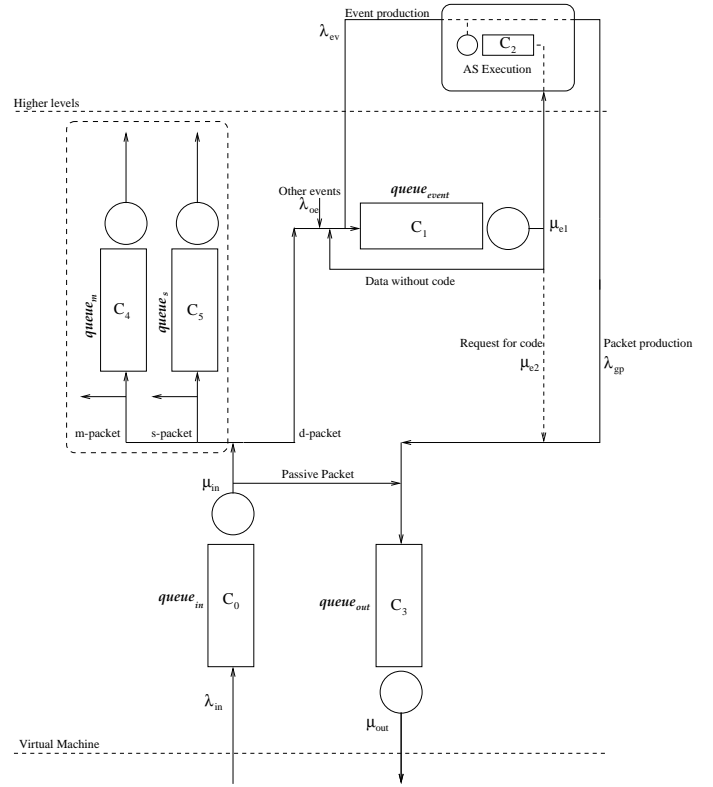


Figure 2: Queuing network of the Active Node

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 requested 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.

Since 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 model concerns only the input, the output, the event and the application service queues. We consider two approaches to model this queuing network:

- 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 rela-

tionship 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 [15] and is solved using the PEPA Workbench [16].

- The second approach consists of using SimJava simulation. SimJava is a discrete event process oriented simulation package developed earlier from 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

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 output 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.

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 $\lambda_{in} = \lambda_{in,a} + \lambda_{in,s}$, 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 standard 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. Regardless of the value of K , the active packets are divided between the different

Active Packets Repartition	Arrival Rates
<i>d_packet</i> : 75%	$\lambda_{ev} = 0.1$
<i>m_packet</i> : 12.5%	$\lambda_{oe} = 0.1$
<i>s_packet</i> : 12.5%	$\lambda_{gp} = 0.1$
Service Rates	Probabilities
$\mu_{in} = 0.5$	$q_p = K$
$\mu_{out} = 0.5$	$q = 0.5$
$\mu_{e1} = 0.5; \mu_{e2} = 0.2$	

Table 1: Input parameters

types of active packets according to the proportions defined in Table 1. In our initial experiments, we assume that all buffers have the same capacity $C_i = 5$, $i = 0, \dots, 3$. The other parameters of the model are summarised in Table 1.

The results obtained are depicted in Figures 3-11. All curves are plotted versus the load of the input queue; we decrease the inter-arrival delay of packets ($1/\lambda_{in}$), thus increasing the load on the node.

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

- Figure 3 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 queue begins to be heavily loaded the throughput tends to be stable, for all values of K . These results are valid for both simulation and analytical modelling (PEPA).

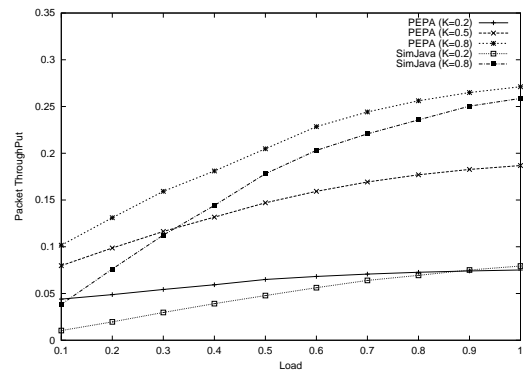


Figure 3: The throughput

- To have an idea of 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. Figure 4 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.

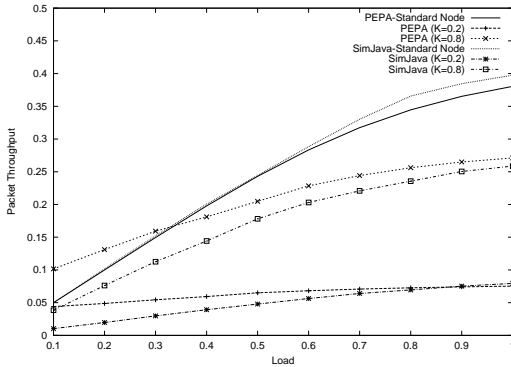


Figure 4: The throughput

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.

- In Figure 5, we see an exponential increase of the loss probability of passive packets as the load increases. This probability increases 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 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 sent to the output queue immediately.

- In Figure 6, we compare the loss probability of passive packets in an active node and the one obtained for a standard node. We consider both

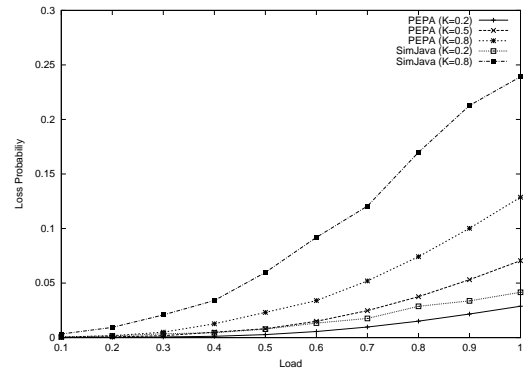


Figure 5: The loss probabilities

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 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.

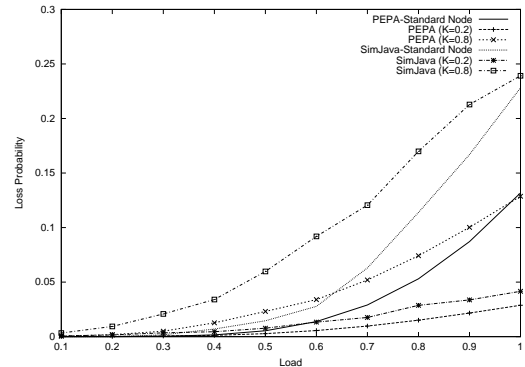


Figure 6: The loss probabilities

3. The last performance measure we are interested in is the latency experienced by the passive packets in an active node.

- Figure 7 shows that for a small buffer capacity, the proportion \bar{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.

Moreover this figure shows that the difference between the latency for a passive packet in an active node with the latency in a standard node is negligible. 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. 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. 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 the latency when $K = 0.8$ is a lower bound for the latency in the standard node.

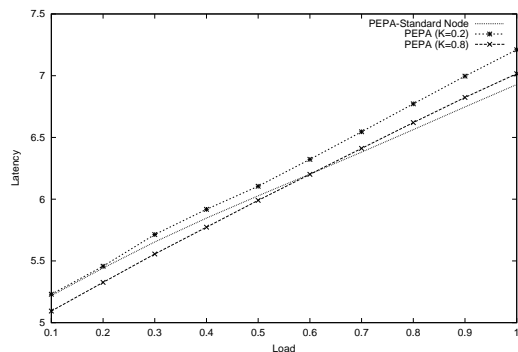


Figure 7: The latency

Impact of the buffers capacity

In these experiments, we investigate the impact of increasing the buffer sizes on both the passive packets' loss and the latency experienced by these packets. For that we consider two cases, $C_i = 10$ and $C_i = 15$, $i = 0, \dots, 3$. As all buffers have the same capacity, in the following, we denote by C the size of these buffers. The results obtained are depicted in Figures 8-11.

1. Figures 8 and 9 show the impact of the buffer capacity on the passive packets' loss.

- In Figure 8, we compare the loss probability of passive packets for the three buffer sizes

$C = 5, 10, 15$. The measures are made for a proportion of passive packets in the node $K = 0.2$, that is when the traffic of standard packets is very perturbed. This figure shows first that the loss probability increases exponentially as the load increases for all buffer capacities. Moreover this probability decreases as the buffers capacity increases.

However, we can notice that the difference between the loss probability when $C = 10$ and $C = 15$ is negligible for a load between 0.1 and 0.7. This difference starts to become significant as the load becomes greater than 0.7, 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 probability will perhaps continue to decrease. However, it seems likely that we will reach a buffer capacity at which this probability will no longer significantly decrease.

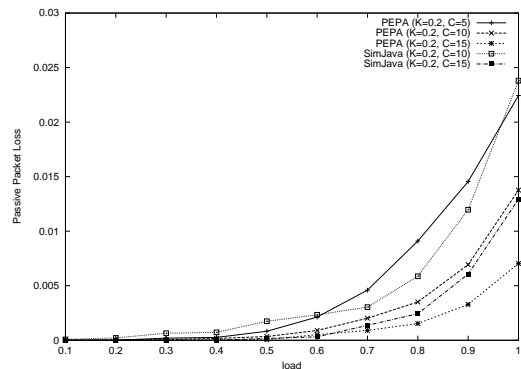


Figure 8: The loss probabilities

- In Figure 9 we compare the loss probability in a standard node with the one in an active node. This comparison is done for a buffers' capacity $C = 10$ and different values of K : 0.2, 0.5, 0.8. Figure 9 confirms the results we have obtained in Figure 6 for $C = 5$ as the loss probability in a standard node can be bounded by the loss probability 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 6, but 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 $C = 10$.

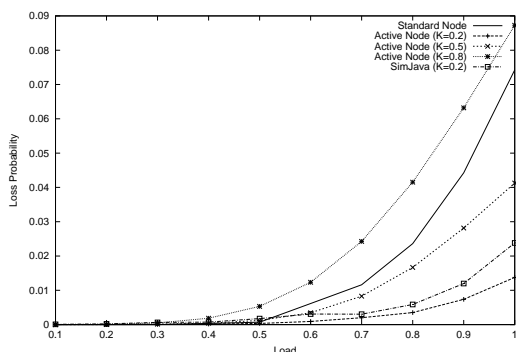


Figure 9: Standard node *versus* active node

2. In the following, we show using Figures 10 and 11 the impact of the buffer capacity on the latency experienced by a standard packet in an active node.

- In Figure 10, we compare the latency obtained for the different buffer sizes $C = 5, 10$ and 15 . The proportion of active packets is $\bar{K} = 0.8$, that is the standard packets traffic is very perturbed. This figure shows that the latency increases as the load increases and this for all values of the buffers' capacity. Moreover this latency increases proportionally to the buffers' sizes. We can notice that the difference between the latency when $C = 15$ and $C = 10$ is smaller than the difference between the latency when $C = 10$ and $C = 5$.

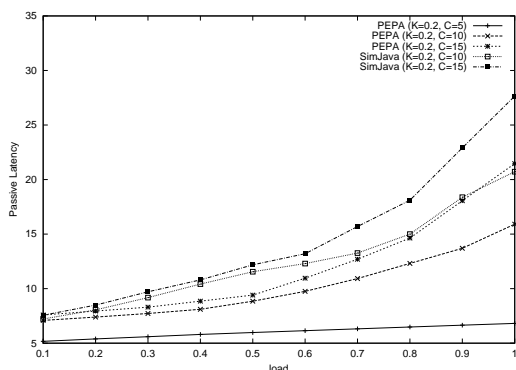


Figure 10: The latency

- In Figure 11, we compare the latency experienced by a standard packet in both types of nodes. We consider buffers' capacity $C = 10$ and different values of K : 0.2, 0.5, 0.8.

Unlike for $C = 5$ (Figure 7) where the impact of the proportion of passive packets in the node is not really significant, Figure 11 shows that for $C = 10$, K has a more important impact on the latency experienced by a passive packet. Moreover, as for the

loss probability, the latency in the standard node is clearly bounded by the latency when $K = 0.5$ (lower bound) and $K = 0.8$ (upper bound). As for the loss probability we may expect to refine these bounds, in particular the upper bound, the lower bound being very close to the standard node latency.

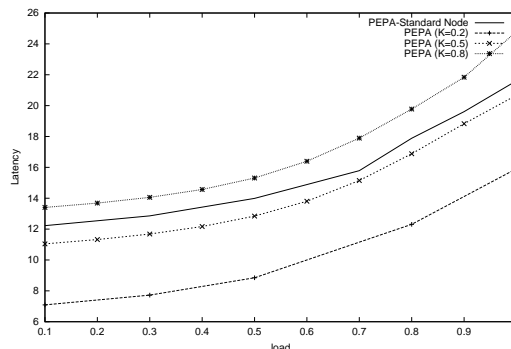


Figure 11: Standard node *versus* active node

Further experiments have been done investigating the impact of varying the arrival rate λ_{gp} of the packets resulting from the execution of the application services. The results obtained confirm the results we have seen so far. These results are not presented here because of the paper size constraint.

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 probability and the latency experienced by the standard packets in the node. Different buffer sizes have been considered. For the loss probability, 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.

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 framework ARFanet is at its final stage. Further work consists of testing the platform's performance and comparing the results obtained with those presented in this paper.

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

References

- [1] D. Tennenhouse, J. Smith, D. Sincoskie, D. Wetherall, G. Minden, *A survey of active network research*, IEEE Communications Magazine, 35(1):80-86, 1997.
- [2] K. Calvert (ed.), *Architectural framework for active networks*, Technical report, AN Architecture Working Group, July 1998.
- [3] H. Hartman, L. Peterson, A. Bavier, P. Bigot, P. Bridges, B. Montz, R. Piltz, T. Proebsting, O. Spatscheck, *Joust: A platform for liquid software*, In: IEEE Computer Magazine, pp. 50-56, April 1999.
- [4] S. Merugu, S. Bhattacharjee, E. Zegura, K. Calvert, *Bowman: A node OS for active networks*, Proc. of IEEE Infocom 2000, Tel Aviv, Israel, March 2000.
- [5] P. Tullmann, H. Hibler, J. Lepreau, *Janos: A Java-oriented OS for active networks*, IEEE Journal on Selected Areas of Communication, Vol. 19, N. 3, March 2001.
- [6] M. Bouzeghoub, L. Kloul, A. Mokhtari, *A new active network framework based on active rules*, Technical Report, N. 2002/21 PRiSM, Université de Versailles 2002.
- [7] R. McNab, F.W. Howell, *Using Java for discrete event simulation*, In Proc. of the Twelfth UK Computer and Telecommunications Performance Engineering Workshop (UKPEW), Edinburgh, 1996.
- [8] F. Howell, R. McNab, "SimJava: a discrete event simulation package for Java with applications in computer systems modelling", In proc. First Int. 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>
- [9] J. Hillston, *A compositional approach to performance modelling*, PhD. The University of Edinburgh, 1994.
- [10] D.R.W. Holton *A PEPA specification of an industrial production cell*, Proceedings of the Third Int. Workshop on Process Algebras and Performance Modelling. Special Issue of The Computer Journal, 38(7), December 1995.
- [11] S. Gilmore, J. Hillston, D.R.W. Holton, M. Rettelbach, *Specifications in stochastic process algebra for a robot control problem*, International Journal of Production Research, 34(4):1065-1080, 1996.
- [12] J. Hillston, L. Kloul, *Performance investigation of an on-line auction system*, Concurrency and Computation: Practice and Experience 13(1): 23-41, Wiley & Sons 2001.
- [13] J.-M. Fourneau, L. Kloul, F. Valois, *Performance modelling of hierarchical cellular networks using PEPA*, Performance Evaluation 50(2/3): 83-99, Elsevier Science 2002.
- [14] S. Gilmore, L. Kloul, *A Unified tool for performance modelling and prediction*, The 22nd Int. Conf. on Computer Safety, Reliability and Security, LNCS Vol. 2788: 179-192, Springer Verlag, Edinburgh 2003
- [15] J. Hillston, L. Kloul, A. Mokhtari, *Active nodes performance analysis using PEPA*, In: proceedings of the nineteenth UK Performance Engineering Workshop (UKPEW'03), Warwick, UK, July 2003.
- [16] S. Gilmore, J. Hillston, *The PEPA workbench: a tool to support a process algebra-based approach to performance modelling*, In: Proc. of the Seventh Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation, LNCS, Vol. 794: 353-368, Springer-Verlag, Vienna 1994.