Modelling a Network Traffic Probe Over a Multiprocessor Architecture

Luis Zabala, Armando Ferro,
Alberto Pineda and Alejandro Muñoz
University of the Basque Country (UPV/EHU)
Spain

1. Introduction

The need to monitor and analyse data traffic grows with increasing network usage by businesses and domestic users. Disciplines such as security, quality of service analysis, network management, billing and even routing require traffic monitoring and analysis systems with high performance. Thus, the increasing bandwidth in data networks and the amount and variety of network traffic have increased the functional requirements for applications that capture, process or store monitored traffic. Besides, the availability of capture hardware (monitoring cards, taps, etc.) and mass storage solutions at a reasonable cost makes the situation better in the field of network traffic monitoring. For these reasons, several research groups are studying how to monitor heterogeneous network environments, such as wired broadband backbone networks, next generation cellular networks, high-speed access networks or WLAN in campus-like environments. In keeping with this line, our research group NQaS (Networking, Quality and Security) aims to contribute in this challenge and presents theoretical and experimental research to study the behaviour of a probe (Ksensor) that can perform traffic capturing and analysis tasks in Gigabit Ethernet networks. Not only do we intend to progress in the design of traffic analysis systems, but we also want to obtain mathematical models to study the performance of these devices.

The widespread of 1/10 Gigabit Ethernet networks, emphasizes the problems related to system losses which invalidate the results for certain analyses. New Gigabit networks, even at 40 and 100 Gbps, are already being implemented and the problem becomes accentuated. On top of that, commodity systems are not optimized for monitoring [Wang&Liu, 2004] and, as a result, processing resources are often wasted on inefficient tasks. Because of this, new research works have arisen focusing on the development of analysis systems that are able to process all the information carried by actual networks.

Taking all this into account, we would like to develop analytical models that represent traffic monitoring systems in order to provide solutions to the problems mentioned before. Modelling helps to predict the system's performance when it is subjected to a variety of network traffic load conditions. Designers and administrators can identify bottlenecks, deficiencies and key system parameters that impact its performance, and thereby the system can be properly tuned to give the optimal performance. By means of modelling technique, it is possible to draw qualitative and, in many cases, also quantitative conclusions about features related to modelled systems even without having to develop them. The impact of developing costs, which is a determining factor in some cases, can be dramatically reduced by using modelling.

Having this in mind, and considering the experience of our group, we present our original design (Ksensor) that improves system performance, as well as a mathematical model based on a closed queueing network which represents the behaviour of a multiprocessor traffic monitoring and analysis system. Both things are considered together in the validation of the model, where Ksensor is used as well as a testing platform developed by NQaS. All these aspects are presented throughout this chapter.

A number of papers has addressed the issue of modelling traffic monitoring systems. However, there are more related to the hardware and software involved in this type of systems.

Regarding hardware proposals, one of the most relevant was the development of the high-performance DAG capture cards [Cleary et al., 2000] at the University of Waikato (New Zealand). Several research works and projects have made use of these cards for traffic analysis system design. Some other works proposed the use of Network Processors (NP) [Intel, 2002]. Conventional hardware also showed bottlenecks and new input/output architectures were proposed, such as Intel’s CSA (Communication Streaming Architecture).

At the software level, Mogul and Ramakrishnan [Mogul&Ramakrishnan, 1996] identified the most important performance issues on interrupt-driven capture systems. Zero-copy architectures are also remarkable [Zhu et al, 2006]. They try to omit the path followed by packets through the system kernel to the user-level applications, providing a direct access to captured data or mapping memory spaces (mmap). Biswas and Sinha proposed a DMA ring architecture [Biswas&Sinha, 2006] shared by user and kernel levels. Luca Deri suggests a passive traffic monitoring system over general purpose hardware at Gbps speeds (nProbe). Deri has also suggested improvements for the capture subsystem of GNU/Linux, such as a driver-level ring [Deri, 2004], and a user-level library, nCap [Deri, 2005a]. Recently, Deri has proposed a method for speeding up network analysis applications running on Virtual Machines [Cardigliano, 2011], and has presented a framework [Fusco&Deri, 2011] that can be exploited to design and implement this kind of applications.

Other proposals focus on parallel systems. Varenni et al. described the logic architecture of a multiprocessor monitoring system based on a circular capture buffer [Varenni et al.,2003] and designed an SMP driver for DAG cards. We must also remark the KNET module [Lemoine et al., 2003], a packet classifying system at the NIC to provide independent per connection queues for processors. In addition, Schneider and Wallerich studied the performance challenges over general purpose architectures and described a methodology [Schneider, 2007] for evaluating and selecting the ideal hardware/software in order to monitor high-speed networks.

Apart from the different proposals about architectures for capture and analysis systems, there are analytical studies which aim at the performance evaluation of these computer systems. Among them, we want to underline the works done by the group led by Salah

[Salah, 2006][Salah et al., 2007]. They analyse the performance of the capturing system considering CPU consumptions in a model based on queuing theory. Their last contributions explain the evolution of their models towards applications like Snort or PC software routers. Another work in the same line was developed by Wu [Wu et al., 2007], where a mathematical model based on the 'token bucket' algorithm characterized Linux packet reception process.

We also have identified more complex models whose application to traffic capturing and analysis systems can be very beneficial. They are models based on queuing systems with vacations. In this field, we want to underline the contributions from Lee [Lee, 1989], Takagi [Takagi, 1994, 1995] and Fiems [Fiems, 2004].

Most of the previous approaches are for single processor architectures. However, it is clear interest in the construction of analytical models for multiprocessor architectures, in order to evaluate their performance. This paper contributes in this sense from a different point of view, given that the model is based on a closed queueing network. Furthermore, the analytical model and the techniques presented in this paper can be considerably useful not only to model traffic monitoring systems, but also to characterize similarly-behaving queueing systems, particularly those of multiple-stage service. These systems may include intrusion detection systems, network firewalls, routers, etc.

The rest of the chapter is organized as follows: in Section 2 we introduce the framework of our traffic and analysis system called 'Ksensor'. Section 3 presents the analytical model for evaluating the performance of the traffic monitoring system. Section 4 provides details on the analytical solution of the model. Section 5 deals with the validation and obtained results are discussed. Finally, Section 6 remarks the conclusions and future work.

2. Ksensor: Multithreaded kernel-level probe

In a previous work [Muñoz et al., 2007], our research group, NQaS, proposed a design for an architecture able to cope with high-speed traffic monitoring using commodity hardware. This kernel-level framework is called Ksensor and its design is based on the following elements:

- Migration to the kernel which consists in migrating the processing module from user-level to the kernel of the operating system.
- Execution threads defined to take advantage of multiprocessor architectures at kernel-level and solve priority problems. Independent instances are defined for capture and analysis phases. There are as many analysing instances as processors, and as many capturing instances as capturing NICs.
- A single packet queue, shared by all the analysing instances, omitting the filtering module and so saving processing resources for the analysis.

This section explains the main aspects of Ksensor, because of its importance in the validation of the mathematical model which will be explained in a subsequent section.

2.1 Architecture of Ksensor

The kernel-level framework, called Ksensor, intended to exploit the parallelism in QoS algorithms, improving the overall performance.

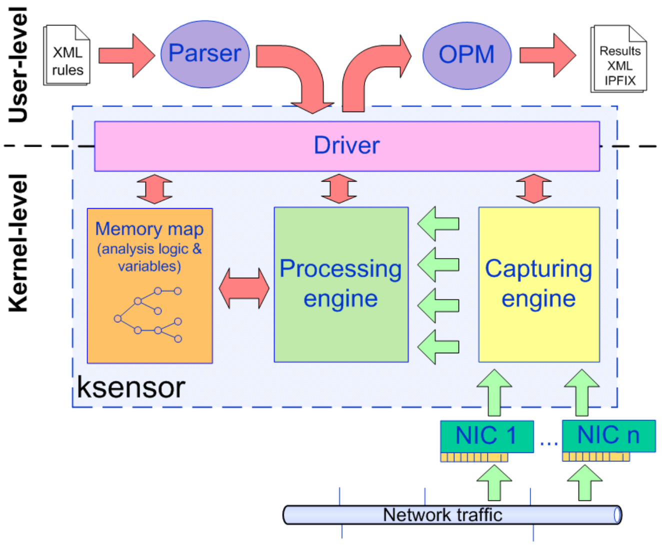

Fig. 1. Architecture of Ksensor.

Fig. 1 shows the architecture of Ksensor. As we can see, only the system configuration (parser) and the result management (Offline Processing Module, OPM) modules are at user-level. Communication between user and Kernel spaces is offered by a module called driver. The figure also shows a module called memory map. This module is shared memory where the analysis logic and some variables are stored.

The definition of execution threads is aimed to take advantage of multiprocessor architectures at kernel-level and solve priority problems, minimizing context and CPU switching. Kernel threads are scheduled at the same level than other processes, so the Kernel's scheduler is responsible for this task.

Ksensor executes two tasks. On one hand, it has to capture network traffic. On the other hand, it has to analyse those captured packets. In order to do that, we define independent instances for capture and analysis phases. Each thread belongs to an execution instance of the system and is always linked with the same processor. All threads share information through the Kernel memory.

In Fig. 2 we can see the multithreaded execution instances in Ksensor. There are as many analysing instances as processors (ksensord#n) and as many capturing instances as capturing NICs (ksoftirqd#n). For example, if the system has two processors, one of them is responsible for capturing packets and analysing some of them and the other one is responsible for analysing packets. This way an analysis task could fill the 100% of one processor's resources if necessary.

The capturing instance takes the packets that the networking subsystem captures and stores them in the packet queue. There is only one packet queue. Processing instances take packets from that queue in order to analyse them.

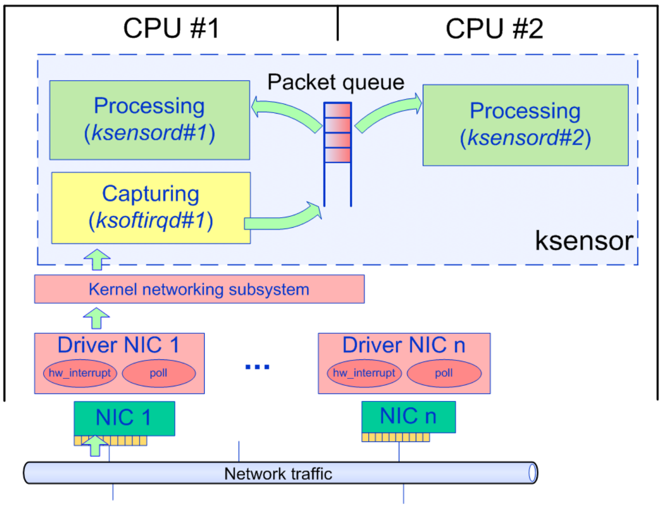

Fig. 2. Multithreaded execution instances in Ksensor.

It does not matter what processing thread analyses a packet because all of them use the same analysis logic. As we said before, there is a shared memory (memory map module) that stores the analysis logic. All the processing threads can access this memory.

2.2 Capturing mechanism in Linux

Ksensor is integrated into the Linux Kernel. In order to capture the packets of the net, Ksensor uses the Kernel networking subsystem. The capturing interface of this subsystem is called NAPI (New API). Nowadays, all the devices have been upgraded to NAPI. Because of that it is important to explain how this interface works [Benvenuti, 2006].

When the first packet arrives to the NIC, it is stored on the card's internal buffer. When the PCI bus is free, the packets are copied from the NIC's buffer to a ring buffer through DMA. The ring buffer is also known as DMA buffer. Once this copy has finished, a hardware interrupt (hardirq) is generated. All of these actions have been executed without consuming any processor's resources.

If the network interface copies a lot of packets in the ring buffer and the Kernel does not take them out, the ring buffer fills up. In this case, unless the interrupts are disabled, another interrupt is generated in order to notify this situation. Then, while the ring buffer is full, the new captured packets will be stored on the NIC's buffer. When this buffer fills up too, the arriving packets will be dropped.

In any case, when the kernel detects the network card interrupt, its handler is executed. In this handler, the NIC driver registers the network interface in an especial list called poll list. This means that this interface has captured packets and needs the Kernel to take them out of the ring buffer. In order to do that, a new software interrupt (softIRQ) is scheduled. Finally, hardIRQs are disabled. From now on, the NIC will not notify new packet arrivals or overload of the ring buffer.

2.3 Network interfaces polling

The softIRQ handler takes out packets from the ring buffer. In Ksensor, after taking out a packet from the ring buffer, the handler stores it in a special queue called packet queue, as we can see in Fig. 2.

The system decides when a softIRQ handler is executed. When its execution starts, the handler polls the first interface in the poll list and starts taking out packets from its ring buffer. In each poll, the softIRQ handler can only pull out packets up to a maximum number called quota. When it reaches the quota it has to poll the next interface in the poll list. If an interface does not have more packets it is deleted from the poll list. Besides, in a softIRQ, the handler can only take out a maximum number of packets called budget. When the handler reaches this maximum, the softIRQ finishes. If there are interfaces left in the poll list, a new softIRQ is scheduled. Furthermore, a softIRQ may take one jiffy (4 ms) at most. If it consumes this time and there are still packets to pull out, the softIRQ finishes and a new one is scheduled.

There is only one poll list in each processor. When the hardIRQ handler is called it registers the network interface in the poll list of the processor that is executing the handler. The softIRQ handler is executed in the same processor. At any given time, a network interface can only be registered in one poll list.

Ksensor has a system to improve the performance in case of congestion. When the packet queue reaches a maximum number of stored packets, this system forces NAPI to stop capturing packets. This means that all the resources of all the processors are dedicated to analysing instances. When the number of packets in the packet queue reaches a fixed threshold value the system starts capturing again.

3. Model for a traffic monitoring system

This section introduces an analytical model which works out some characteristics of network traffic analysis systems. There are several alternatives to model theoretically this type of system. For example, you can use models of queuing theory, Petri nets and, even, mixed models. The ultimate goal is to have a theoretical model that allows us to study the performance of a network traffic analysis system, considering those parameters that are the most representative: throughput, number of processors, analysis load and so on.

We have chosen a theoretical model based on closed queuing networks. It is able to represent accurately the behaviour of a system in charge of analysing network traffic loaded in a multiprocessor architecture. Queuing theory allows us to develop models in order to study the performance of computer's systems [Kobayashi, 1978]. Proposed model consists in a closed queue network where CPU consumptions are related to the service capacity of the queues.

It is worth mentioning that both the flowing traffic and the processing capacity at the nodes are modelled by Poisson arrival rates and exponential service rates. Poisson's distributions are considered to be acceptable for modelling incoming traffic [Barakat et al., 2002]. This assumption can be relaxed to more general processes such as MAPs (Markov Arrival Processes) [Altman et al., 2000], or non homogeneous Poisson processes, but we will keep working with it for simplicity of the analysis. Regarding service rate modelling, although program's code has a quite deterministic behaviour, some randomness is introduced by Poisson incoming traffic, variable length of packets and kernel scheduler uncertainty.

3.1 Description of the model

The proposed queuing network for modelling a traffic monitoring system is showed in Fig. 3. It consists of two parts; the upper one has a set of multi-server queues which represents the processing ability of the traffic analysis system. The lower part models the injection of network traffic with λ rate with a simple queue. The number of packets that are permitted in the closed queue network is fixed and its value is N.

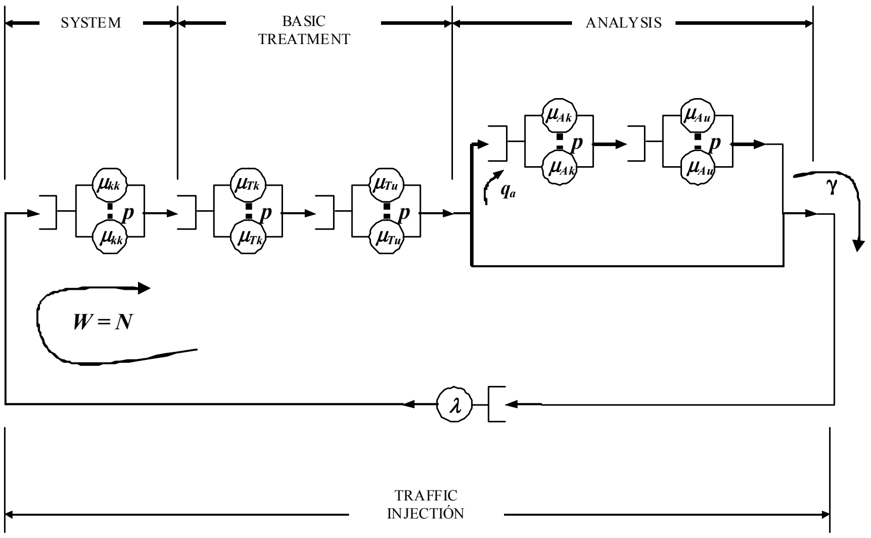

Fig. 3. General model for the traffic analysis system.

Some stages are divided into multiple queues, due to the need to differentiate the processing done in the Kernel and the processing done at user level. Although the process code is usually running on the user level, system calls that require Kernel services are also used.

Four different stages have been distinguished for the closed network, each one with a specific function:

- System stage (system queue): it consists in a queue of μ_{kk} (measured in packets per second) capacity. This stage represents the time spent on the Kernel level of the operating system by the traffic analysis system. It comprises treatments of device controllers and attention paid by kernel to interruptions (hardIRQ and softIRQ) due to packet arrival.
- Basic treatment stage (treatment queues): it is modelled by two queues with μ_{Tk} and μ_{Tu} capacities. This stage represents the amount of time consumed by the system to perform basic treatment to packets captured from the net. This is mainly accomplished by studying control headers of the packets and by determining through a decision tree whether a packet need to be further analysed or not.

- Analysis stage (analysis queues): it is integrated by two queues with μ_{Ak} and μ_{Au}. This stage simulates the analysis treatment that the system does to packets that need further analysis. Not all the packets need to be analysed in this stage. For this reason, a rate called q_a has been defined to represent the proportion of received packet that has to be analysed.
- Traffic injection stage (injection queue): it is a simple queue of λ capacity. This stage simulates the arrival of packets to the system with a λ rate. Since the number of packets in the closed network is fixed to N, the traffic injection queue can be empty. This situation simulates the blocking and new packets will not be introduced on the system.

Each service queue has p servers that represent the p processors of a multiprocessor system. Multiple server representation has been chosen to emphasize the possibility of parallelizing every stage of processing. However, all stages may not be necessarily parallelizable. For example, only one processor can access NIC at the same time, so the packet capturing process will not be parallelizable in different instances.

Another aspect to consider is that packets cannot flow freely in the closed network, because the sum of packets attended in the servers that represent the traffic monitoring system never exceeds the maximum number of processors available. Therefore, we have to assure that, at any time, the maximum number of packets in the upper queues of Fig 3 is not greater than p (the number of processors).

Considering an arrival rate of λ packets per second, the traffic analysis system will be able to keep pace with a part of that traffic, defined as $q\cdot\lambda$. Remaining traffic $((1-q)\cdot\lambda)$ will be lost because the platform is not capable of dealing with all the packets. Captured traffic, $q\cdot\lambda$, goes through the system and basic treatment stages. Nevertheless, all traffic will not be subject of further analysis because of features of the modelled system. For example, a system in charge of calculating QoS parameters of all connections that arrive to a server will discard the packets with other destination address or monitoring systems which use sampling techniques will discard a percentage of packets or intrusion detection will apply further detection techniques only to suspicious packets. Therefore, q_a coefficient has been defined to represent the rate of captured packets liable of being further analysed (analysis stage) than treated only (treatment stage). Thus, $q_a\cdot q\cdot\lambda$ of the initial flow will go through the analysis stage.

3.2 Simplifications of the model

The model presented in Fig. 3 is very general, but if we observe it, some simplifications are possible. Simplifications allow us to group different service rates to identify parameters that may be analysed easily. Among the possible simplifications, we highlight two: one related to CPU consumption and another one, to the equivalent traffic monitoring system.

3.2.1 Model of CPU consumption

This simplification proposes to group all the kernel consumptions in a simple queue, whereas user processes consumptions are represented in a multi-queue. It considers that kernel services are hardly parallelizable.

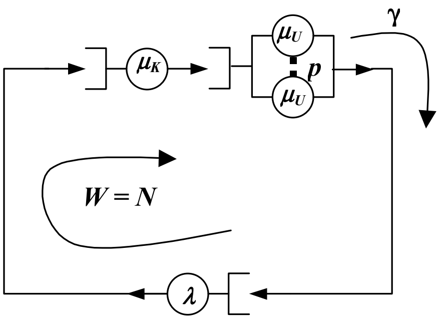

Fig. 4. Model of CPU consumption.

The equivalent service rates can be calculated as follows.

$$\frac{1}{\mu_K} = \frac{1}{\mu_{pk}} + \frac{1}{\mu_{Kk}} = \frac{q_a}{\mu_{Ak}} + \frac{1}{\mu_{Tk}} + \frac{1}{\mu_{Kk}} \tag{1}$$

$$\frac{1}{\mu_U} = \frac{1}{\mu_{pu}} = \frac{q_a}{\mu_{Au}} + \frac{1}{\mu_{Tu}} \tag{2}$$

3.2.2 Model of the equivalent traffic monitoring system

The main feasible simplification preserving the identity of the system is to replace the whole system with an equivalent multi-server queue applying the Norton equivalence [Chandy et al., 1975]. The Norton theorem establishes that in networks with solution in product form, any subnetwork can be replaced by a queue with a state-dependent service capacity. Our theoretical model has exponential service rates in all stages, so applying the Norton equivalence, the new equivalent queue will have a state-dependent service capacity $\mu_{eq}(n,q_a)$.

The simple queue μ_S of the Fig. 5 represents non-parallelizable processes of the system and the multiple queue μ_M represents parallelizable ones.

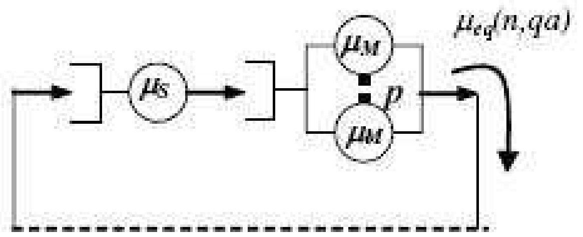

Fig. 5. Traffic monitoring system that Norton equivalence is applied to.

This model adapts perfectly to Ksensor, because we identify a non-parallelizable process that corresponds with the packet capture and parallelizable processes that are related to analysis. Both μ_S and μ_M (in packets per second) can be measured in the laboratory.

4. Analytical study of the model

This section presents the analytical study of the model. It can be directly addressed by analytical calculation, assuming Poisson arrivals and exponential service times. Perhaps the greatest difficulty lies in determining the abstractions that are necessary to adapt the model to the actual characteristics of the traffic monitoring system. Likewise, we propose a method of calculation based on mean value analysis which allows us to solve systems with more elements, where the analytical solution may be more complex to develop.

4.1 Equations of the general model

Viewing the simplifications that have been developed, we might observe that, in the study of this model, a topology is repeated at different levels of abstraction. This topology corresponds with a closed network model with two queues in series; first, a simple one, and second, another one with multiple servers, as shown in Fig. 6. This structure usually occurs in every processing stage. Processing at Kernel level is usually not parallelizable, and therefore, the model is represented as a simple queue. On the other hand, the user processing is usually parallelizable and it is represented by a multiple queue with p servers, being p the number of processor of the platform. The appearance of this topology allows us to define a simple model that we can solve analytically.

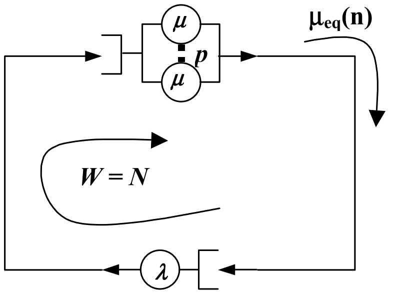

Fig. 6. Closed queue network simplified for the general model.

In order to get the total throughput of the system, first, we calculate the state probabilities for the network, putting N packets in circulation through the closed network, but assuming that the upper multiple queue can have at most p packets being served and the rest waiting in the queue. We also assume that the service capacity in every state of the multiple queue is not proportional to the number of packets. Thus, we will consider μ_i as the service capacity for the state i. The state diagram for this topology is presented in Fig. 7. In this model we are representing the state i of the multiple queue. N packets are flowing through the closed network and we refer to the state i when there are i packets in the multiple queue and the rest, N-i, in the simple queue. The probability of that state is represented as p_i. Finally, the simple queue with rate λ is the packet injection queue.

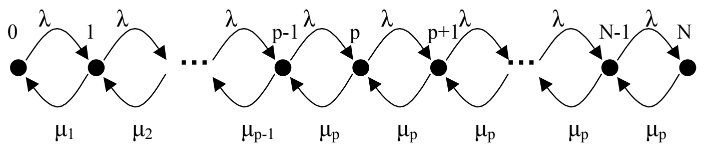

Fig. 7. State diagram for the multiple queue.

It is possible to deduce the balance equations from the diagram of states and, subsequently, the expression of the probability of any state i as a function of the probability of zero state p_0:

$$\forall i = 1,\cdots,p \Rightarrow \left\{ \begin{array}{l} p_0 \cdot \lambda = p_1 \cdot \mu_1 \\ p_1 \cdot \lambda = p_2 \cdot \mu_2 \\ \cdots\cdots\cdots\cdots \\ p_{p-1} \cdot \lambda = p_p \cdot \mu_p \end{array} \right\} \Rightarrow p_i = \frac{\lambda}{\mu_i} \cdot p_{i-1} \tag{3}$$

$$\Rightarrow p_i = \overbrace{\frac{\lambda}{\mu_i} \cdot \frac{\lambda}{\mu_{i-1}} \cdots \frac{\lambda}{\mu_1}}^{\text{i terms}} \cdot p_0 = \frac{\lambda^i}{\prod_{j=1}^{i} \mu_j} \cdot p_0 \tag{4}$$

From this equation, we deduce p_p, the probability of the state p:

$$\Rightarrow p_p = \frac{\lambda^p}{\prod_{j=1}^{p} \mu_j} \cdot p_0 \tag{5}$$

For the states with i>p, their probabilities can be expressed as:

$$\forall i = p+1,\cdots,N \Rightarrow \left\{ \begin{array}{l} p_p \cdot \lambda = p_{p+1} \cdot \mu_p \\ p_{p+1} \cdot \lambda = p_{p+2} \cdot \mu_p \\ \cdots\cdots\cdots\cdots\cdots\cdots \\ p_{N-1} \cdot \lambda = p_N \cdot \mu_p \end{array} \right\} \Rightarrow p_i = p_{i-1} \cdot \frac{\lambda}{\mu_p} \tag{6}$$

$$p_i = \overbrace{\frac{\lambda}{\mu_p} \cdot \frac{\lambda}{\mu_p} \cdots \frac{\lambda}{\mu_p}}^{\text{(i-p) terms}} \cdot p_p = \left(\frac{\lambda}{\mu_p} \right)^{i-p} \cdot p_P \tag{7}$$

From this equation we can also derive the expression of the probability p_N, which is interesting because it indicates the probability of having all the packets in the multiple queue and there is none in the simple queue. This probability defines the blocking probability (P_B) of the simple queue.

$$p_N = P_B = \frac{\lambda^N}{\mu_p^{N-p} \cdot \prod_{j=1}^{p} \mu_j} \cdot p_0 \tag{8}$$

Applying the normalization condition (the sum of all probabilities must be equal to 1), we can obtain the general expression for p_0 and, then, we get every state probabilities.

$$\sum_{i=0}^{N} p_i = 1 = p_0 + \sum_{i=1}^{p} p_i + \sum_{i=p+1}^{N} p_i \tag{9}$$

$$1 = p_0 + p_0 \sum_{i=1}^{p} \frac{\lambda^i}{\prod_{j=1}^{i} \mu_j} + p_0 \frac{\lambda^p}{\prod_{j=1}^{p} \mu_j} \cdot \sum_{i=p+1}^{N} \frac{\lambda^{i-p}}{\mu_p^{i-p}} \tag{10}$$

$$\Rightarrow p_0 = \left(1 + \sum_{i=1}^{p} \frac{\lambda^i}{\prod_{j=1}^{i} \mu_j} + \frac{\lambda^p}{\prod_{j=1}^{p} \mu_j} \cdot \sum_{i=p+1}^{N} \frac{\lambda^{i-p}}{\mu_p^{i-p}} \right)^{-1} \tag{11}$$

Considering equations (8) and (11), we have the following blocking probability p_N.

$$p_N = \frac{\lambda^N \big/ \mu_p^{N-p}}{\left(\prod_{j=1}^{p} \mu_j + \sum_{i=1}^{p} \left(\lambda^i \cdot \prod_{j=i}^{p} \mu_j \right) + \lambda^p \cdot \sum_{i=p+1}^{N} \frac{\lambda^{i-p}}{\mu_p^{i-p}} \right)} \tag{12}$$

P_N is the probability of having N packets in the multiple queue (traffic analysis system queue) of Fig. 6 , so there is not any packet in the injection queue. This situation describes the loss of the system. In order to calculate the throughput γ of the system, (13) is used.

$$\gamma = \lambda \cdot (1 - P_N) \tag{13}$$

Taking into account these expressions, which are valid for the general case, we can develop the equations of the model for some particular cases that will be detailed below: the calculation of the equivalence for the traffic monitoring system and the solution for the closed network with incoming traffic load.

4.2 Calculation of the equivalence for the traffic monitoring system

In general, multiprocessor platforms that implement traffic monitoring systems have certain limitations to parallelize some parts of the processing they do. In particular, Kernel services are not usually parallelizable. This means that, despite having a multiprocessor architecture with p processors that can work in parallel, some services will be performed sequentially and we will lose some of the potential of the platform. For all this, in order to calculate the

Norton equivalence for a traffic monitoring system, one must begin with a model that contains a simple queue and a multi-server queue. This is a particular case of the general model studied before.

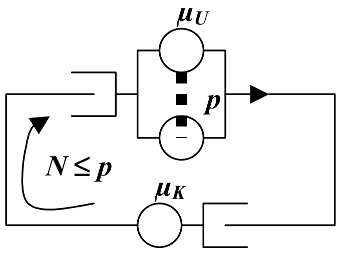

Fig. 8. Equivalence for the traffic monitoring system.

The simple queue with service rate μ_K models non-parallelizable Kernel services, whereas the multiple queue with p servers and service rate μ_U models the system capacity to parallelize certain services. The particularity of this model with regard to the general model is that, at most, only p packets can circulate on the closed network maximum. We are interested in solving this model to work out the equivalent service rate of the traffic monitoring system for every state in the network.

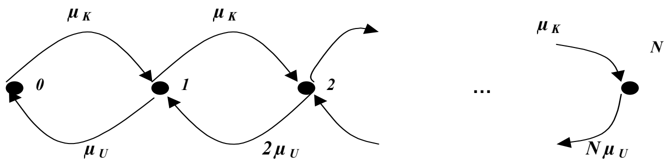

Fig. 9. State diagram for the traffic monitoring system equivalence.

The state diagram makes sense for values of N that are less or equal to the highest number of processors. The service rate of the traffic monitoring system will be different for every value of N and, given that some services are not parallelizable, in general, it does not follow a linear evolution. Following a similar approach to the general case, we can calculate the probability of the highest state, p_N, which is useful to estimate the effective service rate of the equivalence.

$$\left.\begin{array}{c} p_0 \cdot \mu_K = p_1 \cdot \mu_U \\ p_1 \cdot \mu_K = p_2 \cdot 2\mu_U \\ \cdots \\ p_{i-1} \cdot \mu_K = p_i \cdot i \cdot \mu_U \end{array}\right\} \Rightarrow p_i = \frac{\mu_K}{i \cdot \mu_U} \cdot p_{i-1} \tag{14}$$

$$p_i = \frac{\mu_K}{i \cdot \mu_U} \cdot p_{i-1} = \frac{\mu_K^2}{\mu_U^2 \cdot i \cdot (i-1)} \cdot p_{i-2} = \cdots = \frac{\mu_K^i}{\mu_U^i \cdot i!} \cdot p_0 \tag{15}$$

After considering the normalization condition, we can determine the expression for p_N:

$$p_0 + \sum_{i=1}^{N} p_i = 1 = p_0 + \sum_{i=1}^{N} \frac{\mu_K^i}{\mu_U^i \cdot i!} \cdot p_0 = p_0 \cdot \left(1 + \sum_{i=1}^{N} \frac{\rho^i}{i!}\right) \tag{16}$$

$$\Rightarrow p_0 = \frac{1}{\left(1 + \sum_{i=1}^{N} \frac{\rho^i}{i!}\right)} \tag{17}$$

$$p_N = \frac{\mu_K^N}{\mu_U^N \cdot N!} \cdot \frac{1}{\left(1 + \sum_{i=1}^{N} \frac{\rho^i}{i!}\right)} = \frac{\rho^N}{N! + \sum_{i=1}^{N} \frac{N! \cdot \rho^i}{i!}} = \frac{\rho^N}{\sum_{i=0}^{N} \frac{N! \cdot \rho^i}{i!}} \tag{18}$$

Thus, taking into account that the throughput of the closed network is the equivalent service rate, we have the following expression:

$$\mu_{eq}(n) = \mu_K \cdot (1 - p_n) \tag{19}$$

$$\mu_{eq}(n) = \mu_K \cdot \left(1 - \frac{\rho^n}{\sum_{i=0}^{n} \frac{n! \cdot \rho^i}{i!}}\right) \quad // \quad \rho = \frac{\mu_K}{\mu_U} \tag{20}$$

Note that this case is really a particular case of the general case where $\lambda = \mu_K$ and $\mu_i = i \cdot \mu_U$.

4.3 Solution for the closed network model with incoming traffic

The previously explained Norton equivalence takes into consideration the internal problems of the traffic monitoring system related to the non-parallelizable tasks. Now we will complete the model adding the traffic injection queue to the equivalent system calculated before.

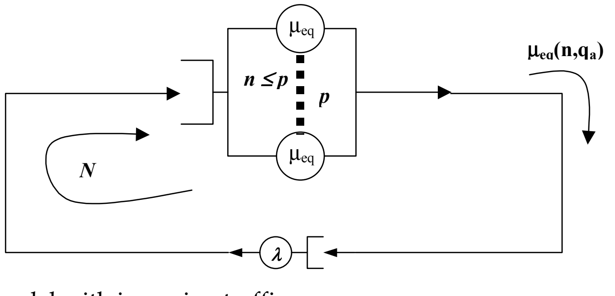

Fig. 10. General model with incoming traffic.

The entire system under traffic load is modelled as a closed network with an upper multiple queue, which is the Norton equivalent queue of the traffic analysis system, and a lower simple queue, simulating the injection of network traffic with rate λ. In this closed network, a finite number N of packets circulate. In general, this number N is greater than p, the number of available processors.

The analytical solution of this model is similar to that proposed for the general model taking into account that the service rates μ_1, μ_2..., μ_p will correspond with the calculation of the Norton equivalent model $\mu_{eq}(n, q_a)$ with values of n from 1 to p. This model allows us to calculate the theoretical throughput of the traffic monitoring system for different loads of network traffic.

$$\gamma = \lambda \cdot (1 - p_N) \tag{21}$$

The value of N will allow us to estimate the system losses. There will be losses when the N packets of the closed network are located in the upper queue. At that time, the traffic injection queue will be empty and, therefore, it will simulate the blocking of the incoming traffic. That will be less likely, the higher the value of N is.

4.4 Mean value analysis

Apart from the analytic solution explained above, we have also considered an iterative method based on the mean value analysis (MVA), in order to simplify the calculations even more. This theorem states that 'when one customer in an N-customer closed system arrives at a service facility he/she observes the rest of the system to be in the equilibrium state for a system with N−1 customers' [Reiser&Lavengerg, 1980]. The application of this theorem to our case requires taking into account the dependencies between some states and others in a complex state diagram, where the state transitions can be also performed with different probabilities, because there are state dependent service rates.

4.4.1 Probability flows between adjacent states

The mean value analysis is based on the iterative dependency between the probability of a certain state with regard to the probabilities of the closest states. The state transitions will not be possible between any two states, they can only occur between adjacent states.

$$p(i,j) = f\big(p(i-1,j), p(i,j-1)\big) \tag{22}$$

It is necessary to do a balance of probability flows between states considering the service rates that are dependent on the state of each queue.

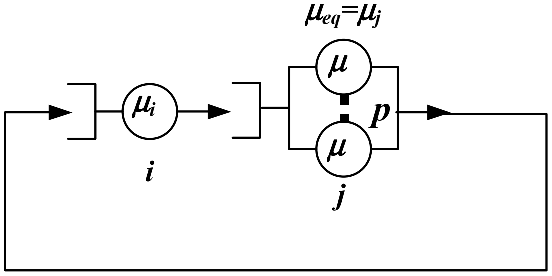

Fig. 11. General model for the closed queue network.

To begin with, we consider the general model for the closed queue network. We call queue i to the simple queue of the model. We assume that this simple queue is in state i and its service rate is μ_i. Likewise, we call queue j to the multi-server queue which is in state j with a state dependent equivalent service rate μ_j. A fixed number of packets (N) are circulating in the closed network, so that there is a dependence between the state i and j.

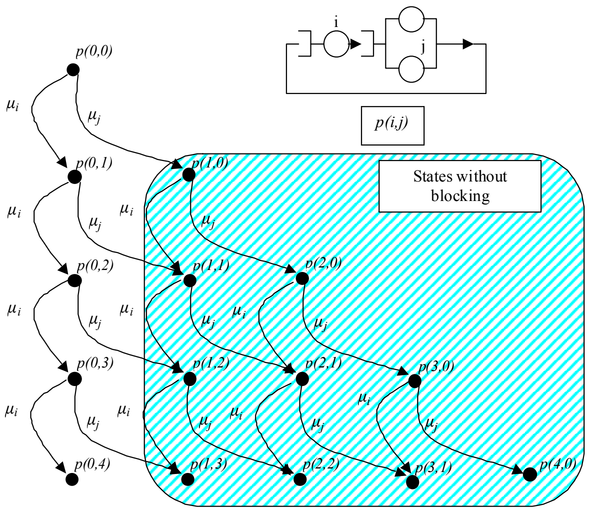

Fig. 12. Probability flows between adjacent states with two processors.

Fig.12 shows the dependencies of the probability of a given state with regard to the closer states in the previous stage with one packet less.

4.4.2 Iterative calculation method

Little's law [Little, 1961] can help us to interpret the relationship between the state probabilities at different stages of the closed queue network.

$$E(T) = \frac{E(n)}{\gamma} \tag{23}$$

This formula is applied to any queue system that is in equilibrium in which there are users who enter the system, consume time to be attended and then leave. In the formula, γ can be understood as the throughput of the system, E(T) as the average time spent in the system and E (n) as the average number of users.

The iterative method applied to the closed queue network is based on solving certain interesting statistics of the network at every stage, using the data obtained in the previous stage. You go from one stage with N packets to the next with N+1 packets, adding one packet to the closed queue network once the system is in stable condition. Knowing the state probability distribution in stage N, we can calculate the average number of users on each server.

$$E(n_i)=\sum_{i=1}^{N} i\cdot p_N(i,N-i) \qquad E(n_j)=\sum_{j=1}^{N} j\cdot p_N(N-j,j) \tag{24}$$

We can calculate every state probability in the stage N as the ratio of the average stay time in this state, $t_N(i,j)$ and the total time for that stage T_{TN}. The total time T_{TN} can be calculated as the sum of all the partial times $t_N(i, j)$ of each state at that stage.

$$p_N(i,j)=\frac{t_N(i,j)}{T_{TOTAL,N}} \tag{25}$$

$$T_{TOTAL,N}=\sum_{i=0}^{N} t_N(i,N-i) \tag{26}$$

If we consider Reiser's theorem [Reiser, 1981], it is possible to set a relation between the state probabilities of a certain state with regard to the ones which are adjacent in the previous stage. In particular, in equilibrium, when we have N packets, the state probability distribution is equal to the distribution at the moment of a new packet arrival at the closed network. In the state diagram of our model, in general, every state depends on two states of the previous stage. We will have the following probability flows:

Transition (i-1,j) $\rightarrow$ (i,j) a new packet arrives at queue i

$$p'_N(i,j)=p_{N-1}(i-1,j) \tag{27}$$

Transition (i,j-1) $\rightarrow$ (i,j) a new packet arrives at queue j

$$p''_N(i,j)=p_{N-1}(i,j-1) \tag{28}$$

Knowing the iterative relations of the probabilities between different stages and basing on Little's formula, we can calculate the average stay time $t_N(i, j)$ in the system in a given state, accumulating the average time in queue i, $t^i_n(i, j)$ and the average time in queue j, $t^j_n(i, j)$.

$$t_N(i,j)=t^i_N(i,j)+t^j_N(i,j) \tag{29}$$

Applying Little's law:

$$t^i_N(i,j)=\frac{E^i_N(i)}{\mu_i(i)}=\frac{p'_N(i,j)\cdot i}{\mu_i(i)}=\frac{p_{N-1}(i-1,j)\cdot i}{\mu_i(i)} \tag{30}$$

$$t^j_N(i,j)=\frac{E^j_N(j)}{\mu_j(j)}=\frac{p''_N(i,j)\cdot j}{\mu_j(j)}=\frac{p_{N-1}(i,j-1)\cdot j}{\mu_j(j)} \tag{31}$$

Considering the probability distribution of the previous stage:

$$t_N(i,j)=\frac{p_{N-1}(i-1,j)\cdot i}{\mu_i(i)}+\frac{p_{N-1}(i,j-1)\cdot j}{\mu_j(j)} \tag{32}$$

Taking into account that, for a given state (i, j), the average stay time of a packet in the queues i and j is given by t_i and t_j respectively, we can express the probability of that state as:

$$\tau_i = \frac{i}{\mu_i(i)} \qquad \tau_j = \frac{j}{\mu_j(j)} \tag{33}$$

$$t_N(i,j) = \frac{p_{N-1}(i-1,j)\cdot i}{\mu_i(i)} + \frac{p_{N-1}(i,j-1)\cdot j}{\mu_j(j)} \tag{34}$$

$$p_N(i,j) = \frac{t_N(i,j)}{T_{TN}} = p_{N-1}(i-1,j)\cdot\frac{\tau_i}{T_{TN}} + p_{N-1}(i,j-1)\cdot\frac{\tau_j}{T_{TN}} \tag{35}$$

Eq. 35 allows us to calculate a certain state probability of the stage with N packets, having the probabilities of the adjacent states in the stage N. Using this equation, we can iteratively calculate the state probability distribution for every stage.

4.4.3 Adjusting losses depending on N

The losses of the traffic monitoring system can be measured assessing the blocking probability of the injection queue. If we consider the general model with an incoming traffic of λ, we can calculate (Eq. 21) the volume of traffic processed by the traffic monitoring system (γ) and also the caused losses (δ).

$$\gamma = \lambda\cdot(1 - p(0,N)) \tag{36}$$

$$\delta = \lambda - \gamma = \lambda\cdot p(0,N) \tag{37}$$

If we look at the evolution of the blocking probability of the injection queue with increasing number of packets N in the closed network, we can see how that probability stage is reduced in each stage. The same conclusion can be derived from Eq. 18.

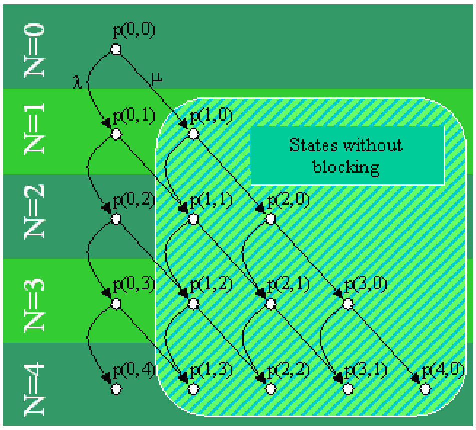

Fig. 13. Evolution of probability flows as a function of N.

A parameter that can be difficult to assess is N, the number of packets that are circulating in the closed network. In general, this parameter depends on specific features of the platform, such as the number of available processors and the ability of the Kernel to accept packets in transit regardless of whether they have processors available at that time.

One conclusion to be drawn from the model, is that it is possible to estimate the value of the parameter N by adjusting the losses that the model has with regard to those which actually occur in a traffic monitoring system.

5. Model validation

This section presents the validation tests to verify the correctness of our analytical model. The aim is to compare theoretical results with those obtained by direct measurement in a real traffic monitoring system, in particular, in the Ksensor prototype developed by NQaS which is integrated into a testing architecture. It is also worth mentioning that, prior to obtaining the theoretical performance results, it is necessary to introduce some input parameters for the model. These initial necessary values will also be extracted from experimental measurements in Ksensor and the testing platform, making use of an appropriate methodology. With all this, we report experimental and analysis results of the traffic monitoring system in terms of two key measures, which are the mean throughput and the CPU utilization. These measures are plotted against incoming packet arrival rate. Finally, we discuss the results obtained.

5.1 Test setup

In this section, we describe the hardware and software setup that we use for our evaluation. Our hardware setup (see Fig. 14) consists of four computers: one for traffic generation (injector), a second one for capturing and analysing the traffic (sensor or Ksensor), a third one for packet reception (receiver) and the last one for managing, configuring and launching the tests (manager). All they are physically connected to the same Gigabit Ethernet switch.

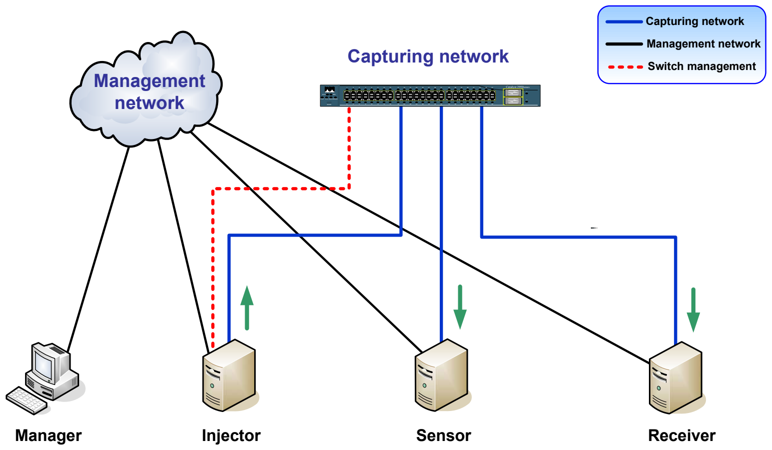

Fig. 14. Hardware setup for validation tests.

However, two virtual networks are distinguished: the first one is the capturing network that connects the elements that play some role during the tests; the second one is the management network which contains the elements that are responsible for the management tasks that can be needed before or after doing tests. The use of two separate networks is necessary, so that the information exchange between the management elements does not interfere with the test results.

The basic idea is to overwhelm Ksensor (sensor) with high traffic generated from the injector. Despite the fact that we do not have 10 Gigabit Ethernet hardware for our tests available, we can achieve our goal of studying the behaviour of the traffic capturing and analysis software at high rates. In addition, we can compare the results with the analytical model and also identify the possible bottlenecks of all analysed systems.

Regarding software, we use a testing architecture [Beaumont et al., 2005] designed by NQaS that allows the automation of tasks like configuration, running and gathering results related to validation tests. The manager, the injector and the sensor that appear in Fig. 14 are part of this testing architecture. They have installed the necessary software to perform the functions of manager, agent, daemon or formatter as we will explain in the next subsection. On the other hand, the receiver is simply the destination of the traffic entered into the network by the injector and it does not have any other purpose.

5.2 Architecture to automatically test a traffic capturing and analysis system

As mentioned previously, in this section, we use a testing architecture for experimental performance measures and, also, to estimate the values of certain input parameters required for the analytical model. It is, therefore, advisable to explain, albeit briefly, the main elements of this platform.

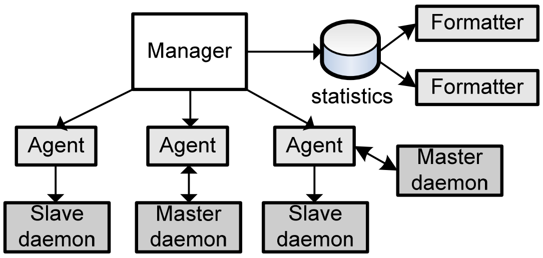

Fig. 15. Logical elements of the testing architecture used in validation tests.

The testing architecture consists of four types of logical elements as Fig. 15 shows. Each of them implements a perfectly defined function:

- Manager is the interface with the user. This element, in the infrastructure shown in Fig. 14, is located on the machine with the same name. It is in charge of managing the rest of the logical elements (agents, daemons and formatters) according to the configuration received from the administrator. After introducing the test setup, it is distributed from the manager to the other elements and the test is launched when the manager sends the start command. At the end of every test, the manager receives and stores the results obtained by the rest of the elements.

- Agents are responsible for attending manager's requests and acting on different devices. Agents are always listening and they have to start and stop the daemons, as well as to collect the performance results. During a test in the infrastructure, one agent is executed in the injector and another one, in the sensor.
- Daemons are in charge of acting on the different physical elements which are involved in each test. Its function can be very variable. For example, the injection of network traffic according to the desired parameterization, the configuration of the capturing buffers, the execution of control programs in the sensor, the acquisition of information or some element's statics, etc. Depending on the relationship with the agent two different types of daemons can be distinguished: master and slave. Master daemons have got some intelligence. The agent will start them but they will indicate when their work has finished. On the other hand, slave daemons do not determine the end of its execution. In each test, to do all the tasks, as many daemons as necessary are executed in the injector and in the sensor.
- Formatters are the programs which select and translate the information stored by the manager to more appropriate formats for its representation. They are executed in the machine called manager, at the end of every test.

5.3 Experimental estimation for certain parameters of the model

In section 3, we have defined an analytical model which functionally responds to a traffic monitoring system. In order to perform an assessment of the model, first we need some values for certain input parameters. We are referring to some service rates that appear in the model based on closed queue networks and are necessary to obtain theoretical performance results. Then we can compare these analytical results with those obtained in the laboratory.

In general, we talk about μ service rates, but, in this subsection, it is easier to talk about mean service times. For this reason, we use the nomenclature based on average processing time in which an average time t_{ij} can be expressed as the inverse of its service rate $1/\mu_{ij}$.

We want to adapt the theoretical model to Ksensor, a real network traffic probe. The best approach is to consider the model of the equivalent traffic monitoring system (see Fig.5) where we distinguish a non-parallelizable process and a parallelizable one. In Ksensor, this separation corresponds with the packet capturing process and the analysis process.

The packet capturing process is not parallelizable because the softIRQ is responsible for the capture and it only runs in one CPU. Fig. 16 shows experimental measurements about average packet capturing times. They have been obtained running tests with Ksensor under different conditions: variable packet injection rate in packets per second and traffic analysis load in number of cycles (null, 1K, 5K or 25K). The inverse of the average softIRQ times shown in Fig. 16 will be the service rate μ_s that appears in the model.

On the other hand, the analysis process is parallelizable in Ksensor. In the same way that softIRQ times have been obtained, we experimentally get average analysis processing times that are shown in Fig. 17. The inverse of the average times shown in Fig.17 will be the service rate μ_M that appears in the multi-queue of the model. It is necessary to comment that, in Fig. 16, the average softIRQ times are not constant. This is because neither all the injected packets are captured by the system, nor all the captured packets are analysed and this causes different computational flow balances.

The values μ_s and μ_M, derived from these experimental measurements, will be taken to the performance evaluation of the model that will be explained later. In addition to the two parameters mentioned, there is another one which is q_a, but it is always $q_a=1$ in our test configuration.

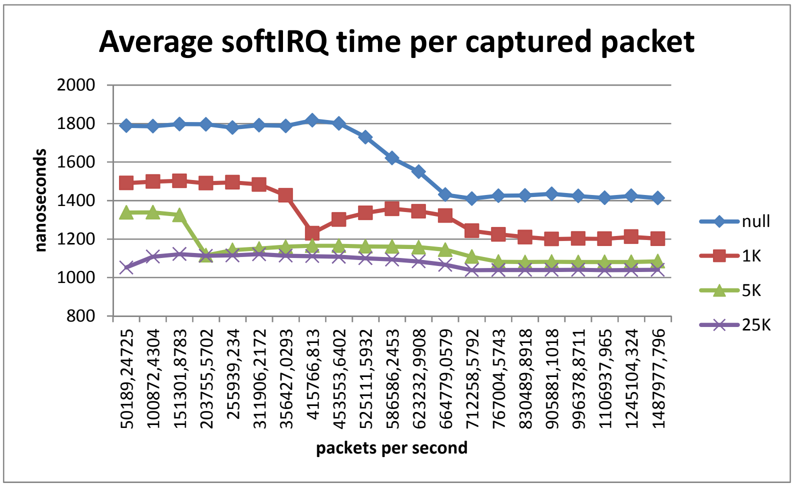

Fig. 16. Average softIRQ per captured packet.

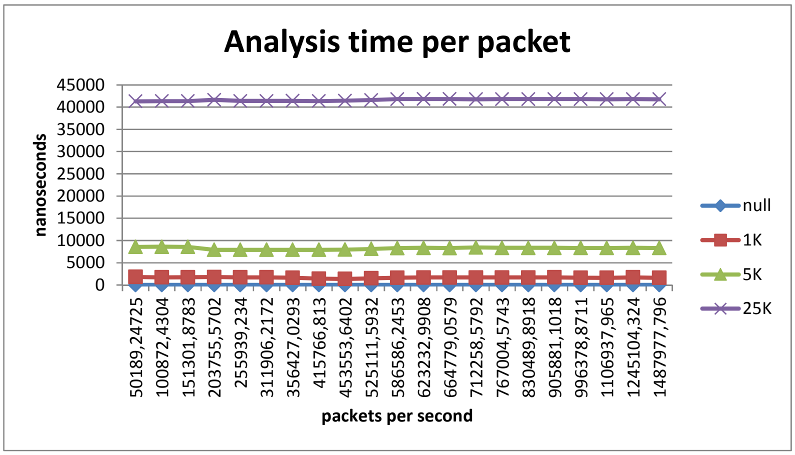

Fig. 17. Analysis time per packet.

5.4 Performance measurements - Evaluation and discussion

The analytical model has been tested with Ksensor under different conditions: packet injection rate (packets per second) varies between 0 and 1.5 million, packet length is 64-1500 bytes and traffic analysis load (at present we simulate QoS algorithm processing times, from 0 to 25000 cycles). The number of processors has been 2 in every test.

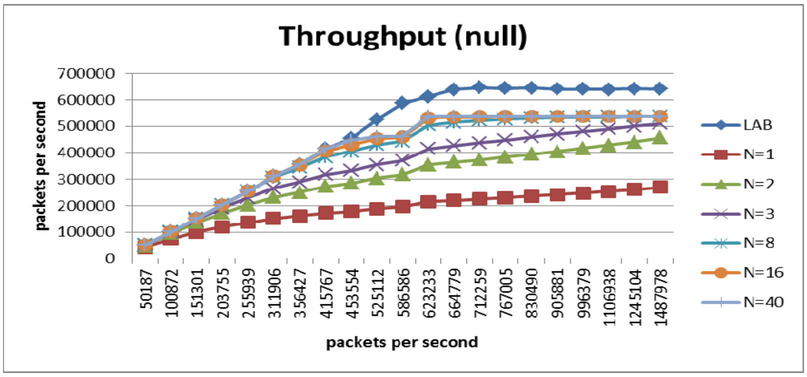

Fig. 18. Theoretical and experimental throughputs without analysis load.

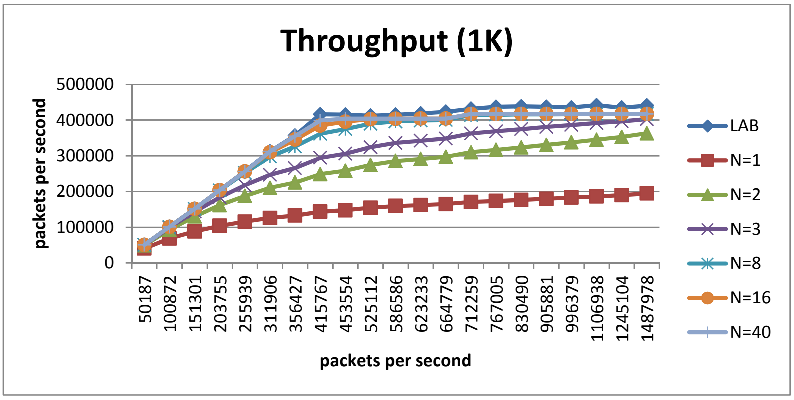

Fig. 19. Theoretical and experimental throughputs with 1Kcycle analysis load.

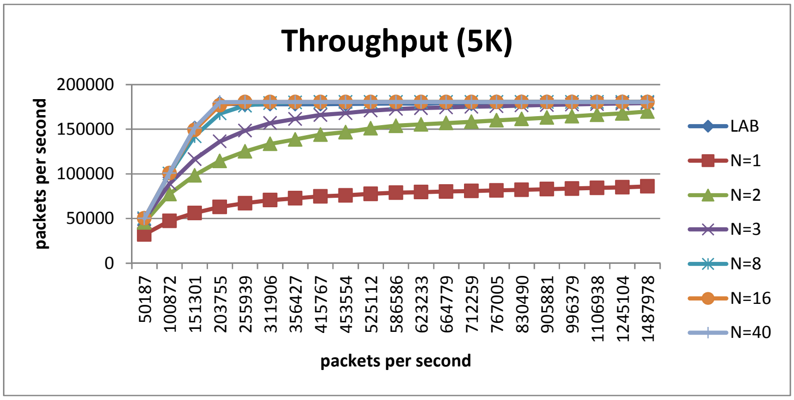

Fig. 20. Theoretical and experimental throughputs with 5Kcycle analysis load.

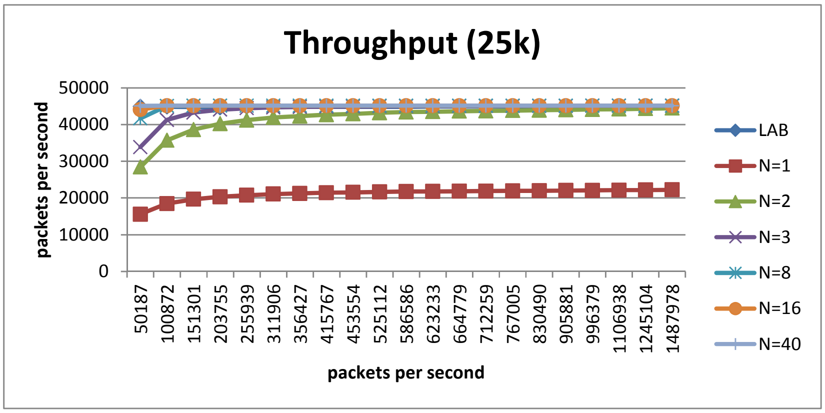

Fig. 21. Theoretical and experimental throughputs with 25Kcycle analysis load.

Fig. 18, Fig. 19, Fig. 20 and Fig. 21 show the comparison between the theoretical model's throughput for different values of N and the real probe's throughput measured experimentally (marked as LAB in the graph). 64 byte-length packets have been used in the lab test and its corresponding service rates in the theoretical calculation. The service rates has been calculated according to the method explained in subsection 5.3.

In all the cases, the throughput grows until a maximum is reached (saturation point). We also observe in these graphs that, with increasing N, the theoretical throughput is close to the real one. It shows, therefore, that the analytical model fits the real system.

6. Conclusion

In this chapter we have presented an analytical model that represents a multiprocessor traffic monitoring system. This model analyses and quantifies the system performance and it can be useful to improve aspects related to hardware and software design. Even, the model can be extended to more complex cases which have not been treated in the laboratory.

Thus, the major contribution of this chapter is the development of a theoretical model based on a closed queuing network that allows to study the behaviour of a multiprocessor network probe. A series of simplifications and adaptations is proposed for the closed network, in order to fit it better to the real system. We obtain the model's analytic solution and we also propose a recurrent calculation method based on the mean value analysis. The model has been validated comparing theoretical results with experimental measures. In the validation process we have made use of a testing architecture that not only has measured the performance, it has also provided values for some necessary input parameters of the mathematical model. Moreover, the architecture helps to setup tests faster as well as to collect and plot results easier. Ksensor, a real probe, is part of the testing architecture and, therefore, it is directly involved in the validation process. As has been seen in the validation section, Ksensor's throughput is acceptably calculated by the model proposed in this chapter. The conclusions obtained have been satisfactory with regard to the behaviour of the model.

This paper has also come in useful to explain the main aspects of Ksensor, a multithreaded kernel-level probe developed by NQaS research group. It is remarkable that this system introduces performance improving design proposals into traffic analysis systems for passive QoS monitoring.

As a future work, we suggest two main lines: the first one is related to Ksensor and it is about a new hardware-centered approach whose objective is to embed our proposals onto programmable network devices like FPGAs. The second research line aims at completing and adapting the model to the real system in a more accurate way. We are already making progress on new mathematical scenarios which can represent, in detail, aspects such as packet capturing process, congestion avoidance mechanisms between capturing and analysis stages, specific analysis algorithms applied in QoS monitoring and packet filtering.

Finally, it is worth mentioning that the test setup, which has been used to validate the model, will be improved acquiring network hardware at 10 Gbps and installing Ksensor over a server with more than two processors. The model will be tested under these new conditions and we hope to obtain satisfactory results, too.

Thus, further work is necessary to analyse this type of systems with a higher precision, compare their results, in certain conditions, better and prevent us from developing high-cost prototypes.

7. References

Altman, E.; Avratchenkov,K. & Barakat, C.. (2000). A stochastic model for TCP/IP with stationary random losses. *ACM SIGCOMM 2000.*

Barakat, C.; Thiran, P.; Iannaccone, G.; Diot, C. & Owezarski, P. (2002). A flow-based model for Internet backbone traffic, *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurement, 2002.*

Beaumont, A.; Fajardo, J.; Ibarrola, E. & Perfecto, C. (2005). Arquitectura de red para la automatización de pruebas. *VI Jornadas de Ingeniería Telemática.,* Vigo, Spain.

Benvenuti, C. (2006). *Understanding Linux Network Internals,* O' Reilly Media.

Biswas, A.; Sinha, P. (2006). Efficient real-time Linux interface for PCI devices: A study on hardening a Network Intrusion Detection System. *SANE 2006,* Delft, The Netherlands.

Cardigliano, A. (2011). Towards wire-speed network monitoring using Virtual Machines. *Master Thesis,* University of Pisa, Italy.

Chandy, K.M.; Herzog, U. & Woo, L.S. (1975). Parametric Analysis of Queueing Networks Learning Techniques, *IBM J. Research and Development*, vol. 19, no. 1, pp. 43-49, January 1975.

Cleary, J.; Donnelly, S.; Graham, I.; McGregor, A. & Pearson, M. (2000). Design principles for accurate passive measurement. *Passive and Active Measurement. PAM 2000,* Hamilton, New Zealand.

Deri, L. (2004). Improving Passive Packet Capture: Beyond Device Polling. *SANE 2004,* Amsterdam, The Netherlands.

Deri, L. (2005). nCap: Wire-speed Packet Capture and Transmission. *E2EMON 2005,* Nice, France.

Fiems, D. (2004). Analysis of discrete-time queueing systems with vacations. *PhD Thesis,* Ghent University, Belgium.

Fusco, F. & Deri, L. (2010). High Speed Network Traffic Analysis with Commodity Multi-core Systems. *Internet Measurement Conference 2010,* Melbourne, Australia.

Intel-CSA. (2002). Communication Streaming Architecture: Reducing the PCI Network Bottleneck.

Kobayashi, H. (1978). *Modeling and Analysis: An Introduction to System Performance Evaluation Methodology,* Ed. Wiley-Interscience, ISBN 0-201-14457-3.

Lee, T. (1989). M M/G/1/N queue with vacation time and limited service discipline. *Performance Evaluation,* vol. 9, no. 3, pp. 181-190.

Lemoine, E.; Pham, C. & Lefèvre, L. (2003). Packet classification in the NIC for improved SMP-based Internet servers. *ICN'04,* Guadeloupe, French Caribbean.

Little, J. D. C. (1961). A proof of the queueing formula: $L=\lambda \cdot W$, *Operations Research*, vol. 9, no. 3, pp. 383-386, 1961.

Mogul, J.C. & Ramakrishnan, K.K. (1996). Eliminating Receive Livelock in an Interrupt-driven Kernel. *USENIX 1996 Annual Technical Conference,* San Diego, California.

Muñoz, A.; Ferro, A.; Liberal, F. & López, J. (2007). A Kernel-Level Monitor over Multiprocessor Architectures for High-Performance Network Analysis with Commodity Hardware. *SensorComm 2007,* Valencia, Spain.

Reiser, M. (1981). Mean value analysis and convolution method for queue-dependent servers in closed queueing networks, *Performance Evaluation*, vol. 1, no. 1, pp. 7-18, January 1981.

Reiser, M. & Lavengerg, S.S. (1980). Mean Value Analysis of Closed Multichain Queueing Networks, *Journal of the ACM*, vol. 27, no. 2, pp. 313-322, April 1980.

Salah, K. (2006). Two analytical models for evaluating performance of Gigabit Ethernet hosts with finite buffer. *AEU - International Journal of Electronics and Communications,* vol. 60, no. 8, pp. 545-556.

Salah, K.; El-Badawi, K. & Haidari, F. (2007). Performance analysis and comparison of interrupt-handling schemes in gigabit networks. *Computer Communications,* vol. 30, no. 17, pp. 3425-3441.

Schneider, F. (2007). Packet Capturing with Contemporary Hardware in 10 Gigabit Ethernet Environments. *Passive and Active Measurement. PAM 2007,* Louvain-la-Neuve, Belgium.

Takagi, H. (1991). *Queueing Analysis, A Foundation of Performance Evaluation Volume 1: Vacation and Priority Systems (Part 1),* North-Holland, Amsterdam, The Netherlands.

Takagi, H. (1994). M/G/1/N Queues with Server Vacations and Exhaustive Service. *Operations Research,* pp. 926-939.

Varenni, G.; Baldi, M.; Degioanni, L. & Risso, F. (2003). Optimizing Packet Capture on Symmetric Multiprocessing Machines. *15th Symposium on Computer Architecture and High Performance Computing,* Sao Paulo, Brazil.

Wang, P. & Liu, Z. (2004). Operating system support for high performance networking, a survey. *The Journal of China Universities of Posts and Telecommunications,* vol. 11, no. 3, pp. 32-42.

Wu, W; Crawford, M. & Bowden, M. (2007). The performance analysis of linux networking - Packet receiving. *Computer Communications,* vol. 30, no. 5, pp. 1044-1057.

Zhu, H.; Liu, T.; Zhou, C. & Chang, G. (2006). Research and Implementation of Zero-Copy Technology Based on Device Driver in Linux. *IMSCCS'06.*