Modeling Packet Processing Time in a Multiprocessor Network Traffic Monitoring System

Luis Zabala, Armando Ferro, and Alberto Pineda
Networking, Quality and Security (NQaS) Research Group
University of the Basque Country (UPV/EHU), Bilbao, Spain

Abstract - Nowadays traffic monitoring is a must for many purposes such as QoS monitoring, IDS, antivirus, network problem detection and so on. Deployment of high speed networks implies problems with this kind of systems to be able to cope with all the traffic in the network. Therefore, it would be interesting to know in advance whether a traffic monitoring system will be able to do its task correctly, or it needs more processing power. This paper presents a mathematical model for multiprocessor traffic monitoring systems that use commodity hardware and general purpose operating systems. In order to establish the different elements of the model we have identified the different stages of the trip of packets from wire to application, as well as the particular behavior of the system and the computational cost for each one of them. With this information, we have built up a model based on a closed queuing network that simulates these different stages of the monitoring system and the possible packet losses. This model allows us to estimate the performance of the monitoring application in terms of throughput.

Keywords: traffic monitoring; performance evaluation; closed queuing network; multi-server system

1 Introduction

The performance of network traffic monitoring systems has a great importance in all kind of network devices. Moreover, there are some applications related to network security and packet analysis which are particularly sensitive to network capturing performance. These include IDS (Intrusion Detection Systems), antivirus, QoS (Quality of Service) analysis systems. In order to achieve Gigabit-order analysis capabilities, most of these systems are forced to use specific hardware and operating systems. This results in more expensive devices and longer development processes.

On the other hand, traditionally, the performance degradation of those systems based on commodity hardware and general purpose operating systems is mostly caused by a design focused on network applications, not on extensive network packets capturing [1]. However, last investigations have demonstrated that packet capture has been improved by enhancing general purpose operating systems for traffic analysis. These results are encouraging because today's commodity hardware offers features and performance that just a few years ago were only provided by costly custom hardware design [2] [3]. Even then, there are still deficiencies to adapt to new architectures such as multithreaded and multicored ones.

In order to continue the improvement of this second kind of systems, it is necessary to identify the design flaws that have an impact on capturing performance. So, it is also important to develop theoretic models and simulation tools that help us in the design and development cycles.

The experience of our research group in the development of a traffic capture and analysis system has encouraged our study of the packet capturing subsystem in general purpose operating systems, especially Linux and its effect on the overall packet analysis performance. In order to improve the traffic monitoring systems' performance we need data about where the packets are lost and why. These data allow us to predict the performance of our system without running real simulations and, therefore, without designing and implementing several prototypes. So, in order to help us in the design and cut down development cycles we have built a model of the different stages of the packet capturing subsystem of Linux Kernel. After obtaining real measurements of the computational costs of each stage we have developed a model that provides us with data about the packet capturing and analysis capabilities, such as the number of packet lost in each stage.

The paper is organized as follows. The background of our work is presented in Section 2. Then, Section 3 describes the analytical model of the packet processing consumptions. The model validation is explained in Section 4. Finally, we conclude the paper in Section 5.

2 Background

In the previous work [4], our research group proposed a prototype of a traffic monitoring system for high speed networks called Ksensor. It works at kernel-level and it is based on Linux. The experimental study of Ksensor has brought some aspects that are worth analyzing.

2.1 Problem Formulation

The aspects mentioned before are related to the time the system takes in order to capture or analyze a packet. We have noticed that this time depends on the network traffic rate and the analysis load which is managed by the traffic monitoring
system. In the laboratory, in order to test Ksensor, we simulate different analysis loads implementing four loops that execute 0, 1000, 5000 and 25000 cycles respectively.

Fig. 1 shows the values of a parameter measured experimentally on Ksensor. In particular, Fig. 1 presents the average softIRQ time per captured packet. This parameter belongs to the capturing stage and, as can be observed, it varies with the injection rate and the analysis loads. It is worth mentioning the three areas that we can observe on the 1 Kcycle analysis load curve. Moreover, in the graphic it can be seen that it takes more time capturing a packet with 0 Kcycle analysis load (null) than with 25 Kcycle analysis load. On the contrary, the average values of the analysis time per packet are similar for all the range of injection traffic rates.

Thus, we intend to make a proposal of a mathematical model which considers the capturing and the analysis stages with their possible variability of CPU consumptions. Certain aspects correspond to our group’s own development. However, we consider that these characteristics can also be extrapolated to more general systems.

Our initial hypotheses take us to consider some packet losses along the capturing stage. Hence, as it happens when duplicate packets or IP fragmentation losses are detected; some activities entail a waste of CPU. In the case that concerns us, due to the shortage of resources or, even, due to control policies of the monitoring application, some packets can be discarded in the capturing stage, producing a CPU consumption that does not result in a captured packet.

### 2.2 Generic Traffic Monitoring Systems

If we consider a typical traffic monitoring system, we usually come across three different phases: capture phase, basic analysis or filtering phase and complex analysis phase. The capture phase can be divided into: hardware processing stage (network interface card), driver processing stage and kernel processing stage.

The basic analysis phase is based on classifying each received packet after studying its features to determine whether the packet must be further analyzed or discarded. This task must be carried out for every captured packet. The real packet processing (e.g. intrusion detection algorithm, QoS algorithm, etc.) is applied in the complex analysis phase but only to those packets that successfully passed the basic analysis phase.

In the next section, we will focus on analyzing the parts of the Linux packet capture subsystem. Since the impact of each parameter depends on its computational costs, we work out the cost of every phase along the capture system and identify which stages would lose packets more likely.

#### 2.3 Packet Losses

As it has been said before, we consider that there are packet losses in the capturing stage. Because of that, it is important to study the capturing system of Linux.

After a thoughtful analysis of the path followed by a packet through Linux operating system, we can identify the points where packet losses are more liable. As a result, these points must be considered in the design of packet capture model. Although Linux is a specific case, the path followed by packets in other operating systems is quite similar to this.

Fig. 2 shows the trip of a packet from its ingress into a Linux end system to its final delivery to the monitoring application, taking into account the possible packet losses along the journey. When a packet arrives to the NIC a hardware interruption (hardIRQ) is generated. The network interface card (NIC) captures packets and copies them in an internal buffer. Then, the DMA (Direct Memory Access) engine of the card, without CPU intervention, is in charge of moving these packets to a special allocation area in the main memory of the system called DMA area. The driver is in charge of attending the hardIRQ. When arrival rate is high, this interruption is generated for a bunch of packets. In order to extract the packets from the DMA area, the driver uses polling mechanism. After any interruption, a software interruption (softIRQ) is scheduled in order to complete the capture of packets. Packets are then moved, during the softIRQ, from the DMA area to another space in the main memory, creating a skb element list. Skb elements are buffers in which the kernel handles network packets [5]. Next, the softIRQ has to move the packet to the monitoring application’s analysis buffer.

Packet losses due to capture deficiencies are represented by $P_{hw}$ and are not very common. Packet losses due to DMA transfer errors are represented by $P_{DMA}$. A $P_{DMA}$ will occur when packet arrival rate is very high and the NIC allocates in
main memory more packets than the system is able to manage, causing an overflow of the DMA’s reserved space in main memory. When the packets are moved from the DMA area to the skb element list there can be eventual losses, which are also very uncommon. These losses are represented by $p_{skb}$. Finally, $p_{enq}$ represents the proportion of packets which cannot be enqueued in the monitoring application’s analysis buffer. Obviously, the length of this queue is limited, so that an overflow of this queue becomes quite likely only at very high arrival rates.

### 3 Packet Process Consumptions Model

This section introduces an analytical model which works out, firstly, the different stages which have been identified in a multiprocessor traffic monitoring system and, secondly, the possible packet losses in the journey from the network card to the analysis application. For a large number of arrivals (heavy traffic conditions), the multiprocessor architecture can be modeled as a closed queuing network [6].

#### 3.1 Description of the Model

The proposal for modeling packet process consumptions of a traffic monitoring system is showed in Fig. 3. It consists in a closed queuing network where computer consumptions are related to the service capacity of the queues. Two parts can be distinguished; the upper one has a set of multi-server queues which represents the traffic monitoring system with its different stages (capture, filtering and analysis), processing abilities ($\mu_{C1}$, $\mu_{F}$, $\mu_{A}$) and possible packet losses ($p_{hw, dma}$, $p_{skb}$, $p_{enq}$). The part on the left models the injection of network traffic with rate $\lambda$. According to some rules, captured packets are filtered and, finally, only selected packets are analyzed in the next stage. This stage represents the amount of time spent on this basic treatment.

- **Capture stage**: it represents the functionalities provided by the operating system which is responsible for capturing packets and moving them from the NIC to the memory of the analysis application. It comprises treatments of device controllers and attention paid by kernel to interruptions (hardIRQ and softIRQ) due to packet arrival. This stage is divided into three multi-server queues with rates $\mu_{C1}$, $\mu_{C2}$, $\mu_{C3}$ (measured in packets per second) capacity, due to the need to differentiate the packet losses inside it. For this, $p_{enq}$ is defined as the probability of having every ring buffer descriptor full and $p_{enq}$ is defined as the probability of not enqueuing packets to the analysis buffer.

- **Filtering stage**: it is modeled by a multi-server queue with rate $\mu_{F}$. According to some rules, captured packets are filtered and, finally, only selected packets are analyzed in the next stage. This stage represents the amount of time spent on this basic treatment.

- **Analysis stage**: it is integrated by a multi-server queue with rate $\mu_{A}$. It simulates the complex analysis treatment that the system does to packets that need further analysis. As not all the packets need to be processed in this stage, a rate called $q_{a}$ indicates the proportion of the received packets that has to be analyzed.

- **Traffic injection stage**: it is a simple queue of $\lambda$ rate. This stage simulates the arrival of packets to the system with $\lambda$ 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. The model also considers the possibility of losing packets due to deficiencies at NIC or DMA’s transfer errors with $p_{hw, dma}$ probability. However, since it is generally recognized that they are very uncommon, those losses will be negligible and $p_{hw, dma}=0$ for solving the model.

![Figure 3. Model based on closed queuing network for packet process consumptions](image)
We assume that the number of processors of the system is \( K \) and each stage can be served by a different number of processors, having \( K_c \) available processors for capturing, \( K_f \) for filtering and \( K_a \) for analysis being \( K_c \leq K, K_f \leq K \) and \( K_a \leq K \). So, the parallelizing level can be different in each phase. 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.

### 3.2 Simplifications of the Model

The model presented in Fig. 3 is very general, but some simplifications are possible. First, it is worth mentioning that both, the flowing traffic and the processing capacity at the nodes, are modeled by Poisson arrival rates and exponential service rates. Poisson’s distributions are considered to be acceptable for certain types of network traffic, for instance, modeling the voice traffic, as explained in [7]. This assumption can be relaxed to more general processes such as MAPs (Markov Arrival Processes) [8] or non-homogeneous Poisson processes. However, for some other types of traffic such as Ethernet network, packet arrivals do not follow a Poisson process but are rather bursty [9]. The case we are dealing with is slightly different because the packet arrival is not directly the traffic of the Ethernet network, but it is the incoming packets from the network card’s buffer to the kernel memory area via DMA. Regarding service rate modeling, although program’s code has a quite deterministic behavior, some randomness is introduced by Poisson incoming traffic, variable length of packets and kernel scheduler uncertainty. An analytical solution can become unmanageable when considering non-Poisson arrivals, even presuming general service times. Despite of these limitations, we will keep working with these assumptions for simplicity of the analysis and, as will be demonstrated in Section 4, the results obtained from our models were closely matching to results obtained from real experimental measurements.

Apart from that, the main feasible simplification preserving the identity of the system is to replace the whole traffic monitoring system with its Norton equivalent [10]. 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 rate \( \mu_{eq,TMS}(n) \).

Therefore, with the aim to obtain the simplified model of Fig. 4, we apply the calculation of the Norton equivalent to the general model of Fig. 3 several times as we will see later.

In the study of this model, we observe that the same topology is repeated at different levels of abstraction. This topology corresponds with a closed network model with two multi-server queues where the output flow of the first queue goes to the second one with a probability of \( 1-p \), whereas it comes back to the first one with a probability of \( p \), as shown in Fig. 5. This structure usually occurs in every processing stage.

![Figure 5. Topology repeated at different levels of abstraction.](image)

### 3.3 Equations of the Repeated Topology

In order to get the Norton equivalent of the traffic monitoring system, first, we calculate the state probabilities for the repeated topology (see Fig. 5), putting \( N \) packets in circulation through the closed network, but assuming that the total system can have at most \( K \) packets being served and the rest waiting in the queue. We also take into account the limitation of \( K_c, K_f \) and \( K_a \) processors in each stage.

The state diagram for this topology is presented in Fig. 6. In this model we are representing the state \( i \) of the multi-server queue on the left of Fig. 5 and we will consider \( \mu_i \) as the service capacity for the state \( i \). \( N \) packets are flowing through the closed network and when there are \( i \) packets in the multi-server queue on the left, the rest, \( N-i \), are in the multi-server queue on the right. The probability of that state \( i \) is represented as \( p_i(i) \). Finally, the output of the multi-server queue with rate \( \mu_2(n_2) \) is the input of the multi-server queue with rate \( \mu_i(n_i) \).

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 state zero \( p_0(0) \).

![Figure 6. State diagram for the multiple queue with \( K_c \) servers (\( N \leq K \)).](image)
The term \( p_s(n) \) indicates the probability of having \( n \) packets at queue 1 being \( N \) the number of packets in the closed network. For \( N \leq K \) (being \( K \) the total number of processors):

\[
\begin{align*}
  p_s(0) = p_s(N) = 1/s_N \\
  p_s(i) = \frac{\mu_s(N-i)}{\lambda} p_s(i) + \mu_s(i) p_s(i+1) \\
  \cdots \\
  p_s(N) = \mu_s(N) p_s(N)
\end{align*}
\]

For every value of \( N \), we calculate the throughput of the closed network and it will be the Norton equivalent which we want.

\[
\begin{align*}
  \mu_{eq}(N=1) &= \sum_{i=1}^{N} \mu_s(i) \cdot p_s(i) = \mu_s(1) \cdot p_s(1) \\
  \mu_{eq}(N=2) &= \sum_{i=1}^{N} \mu_s(i) \cdot p_s(i) = \mu_s(1) \cdot p_s(1) + \mu_s(2) \cdot p_s(2) \\
  \cdots \\
  \mu_{eq}(N) &= \sum_{i=1}^{N} \mu_s(i) \cdot p_s(i) = \mu_s(1) \cdot p_s(1) + \mu_s(2) \cdot p_s(2) + \cdots + \mu_s(N) \cdot p_s(N)
\end{align*}
\]

Taking into account these expressions, we can develop the equations of the whole model as will be detailed below.

### 3.4 Calculation of the Norton equivalent of the traffic monitoring system with losses

In order to calculate the Norton equivalence of the traffic monitoring system with losses, it is necessary to apply equations (1) and (2) several times.

First we compute the Norton equilibrium for the filtering and the analysis stages. Then a second Norton equivalent is computed with the result of the first Norton equivalent and the multi-server queue with rate \( \mu_{C3} \). Later, a third Norton equivalent is computed with the result of the second Norton equivalent and the multi-server queue with rate \( \mu_{C2} \). Finally, a fourth Norton equivalent is computed with the result of the third Norton equivalent and the multi-server queue with rate \( \mu_{C1} \). This last result is the Norton equivalent of the traffic monitoring system. The service rate of the traffic monitoring system will be different for every value of \( N \), i.e. it will be a state-dependent service rate.

For the calculation of the Norton equivalence, it must be remembered that the state diagram makes sense for values of \( N \) that are less or equal to the highest number of processors.

### 3.5 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 number of available processors. Now we complete the model adding the traffic injection queue to the equivalent system calculated before and the model of the entire system with incoming traffic corresponds with Fig. 4. Hence, the entire system under traffic load is modeled as a closed network with an upper queue, which is the Norton equivalent of the traffic monitoring system, and a simple queue on the left of the diagram, simulating the injection of network traffic with \( \lambda \) rate. In this closed network, a finite number \( N \) of packets circulates. This number \( N \) is greater than \( K \), the number of available processors.

The analytical solution of this model is similar to that proposed for the repeated topology, taking into account the following: the arrival rate is \( \lambda \) and it is not state-dependent; the service rates \( \mu_1, \mu_2, \ldots, \mu_p \) correspond with the calculation of the Norton equivalent of the traffic monitoring system, thus \( \mu_s(\mu_{eq,TMS}(n)) \) with values of \( n \) from 1 to \( p \); for states \( n \) with \( K < n \leq N \), \( \mu_s(\mu_{eq,TMS}(K)) \).

![Figure 7. State diagram for the traffic monitoring system with incoming traffic (N>K).](image)

\[ \gamma = \lambda \cdot (1 - p(N)) \]

### 4 Model Validation

This section explains the validation tests of the analytical model presented in this paper. The aim is to compare theoretical results with those obtained by experimental tests over a real traffic monitoring system. Although the model is proposed for a generic traffic monitoring system, in this section, the model is adapted and validated for the prototype called Ksensor. We expect to continue with the model validation over other platforms in the near future. It is also worth mentioning that some initial values are needed to assess
the theoretical model. They have been extracted from experimental measurements of Ksensor too.

4.1 Test Setup

Our hardware setup consists, as Fig. 8 shows, of four computers: one for traffic generation (injector), a second one for capturing and analyzing the traffic (Ksensor), a third one for packet reception (receiver) and the last one (manager) for managing, configuring and launching the tests. All they are physically connected to the same Gigabit Ethernet switch.

The basic idea is to overwhelm the system under test, Ksensor, with high traffic generated from the injector. In order to inject traffic bursts, we have installed an Endace 4.3GE DAG card [11] on the injector. Regarding software, we use a testing architecture designed by our research group [12]. The computer called manager is in charge of doing all the necessary tasks in order to automate the tests and measure the performance metrics of interest.

![Figure 8. Hardware setup for validation tests.](image)

The prototype Ksensor is a two-processor probe. One of the CPUs is responsible for capturing all the possible packets and analyzing some of them, whereas the other CPU is responsible only for analyzing packets.

4.2 Experimental Estimation for Certain Input Parameters of the Model

The model explained in Section 3 requires some input parameters such as $\mu_{C_1}, \mu_{C_2}, \mu_{C_3}, \mu_F, \mu_A, p_{enq}, p_{ skb}, q_a$. We are referring to the service rates and probabilities that appear in the model (see Fig. 3) and are needed to obtain theoretical results.

At the end of every test, the manager (see Fig. 8) collects the measurements from Ksensor and the injector. Based on some of these experimental measurements (e.g. mean time consumed by a packet in every stage identified in the model, number of captured packets in softIRQ), we can estimate the model’s input parameters. For example, the service rates $\mu_{C_1}, \mu_{C_2}, \mu_{C_3}, \mu_F$ and $\mu_A$ are computed as the inverses of the mean times experimentally obtained.

We have made tests varying the packet injection rate between 49487 and 1488000 packets per second, with packets of 40 bytes mean length and variable analysis loads (null, 1K, 5K, 25K) and we have observed that both the service rates and the probabilities are dependent on the injection rate and the analysis load. The exception is the probability $q_a$ because it is set before the test starts.

4.3 Performance Measurement Evaluation

After calculating the necessary values of the input parameters, the analytical expressions of Section 3 are assessed numerically. Table I and Fig. 9 illustrate this process showing, as an example, some numerical values for the case of 1K analysis load.

First, we have some experimental measurements obtained from Ksensor and the injector (e.g. the injection rate $\lambda$ and the number of captured packets in softIRQ in Table I). Based on those measurements, we calculate the needed input parameters like the probability $p_{enq}$ (see Table I).

<table>
<thead>
<tr>
<th>Injection Rate (pps)</th>
<th>Captured Packets</th>
<th>Test seconds</th>
<th>$p_{enq}$</th>
<th>$\mu_{TMS}(1)$ (pps)</th>
<th>$\mu_{TMS}(2)$ (pps)</th>
</tr>
</thead>
<tbody>
<tr>
<td>49487</td>
<td>11924238</td>
<td>240.97</td>
<td>0.00</td>
<td>210690</td>
<td>385326</td>
</tr>
<tr>
<td>151298</td>
<td>36457717</td>
<td>240.97</td>
<td>0.00</td>
<td>210195</td>
<td>382273</td>
</tr>
<tr>
<td>255954</td>
<td>61678719</td>
<td>240.97</td>
<td>0.00</td>
<td>210546</td>
<td>383161</td>
</tr>
<tr>
<td>356470</td>
<td>85898013</td>
<td>240.98</td>
<td>0.00</td>
<td>213581</td>
<td>390861</td>
</tr>
<tr>
<td>453538</td>
<td>100799835</td>
<td>240.97</td>
<td>0.08</td>
<td>219462</td>
<td>399052</td>
</tr>
<tr>
<td>586642</td>
<td>100799835</td>
<td>240.97</td>
<td>0.29</td>
<td>216803</td>
<td>403453</td>
</tr>
<tr>
<td>664903</td>
<td>100037774</td>
<td>240.97</td>
<td>0.36</td>
<td>218532</td>
<td>405820</td>
</tr>
<tr>
<td>906022</td>
<td>105326168</td>
<td>240.97</td>
<td>0.51</td>
<td>224472</td>
<td>417279</td>
</tr>
<tr>
<td>1244992</td>
<td>104778112</td>
<td>240.97</td>
<td>0.65</td>
<td>223863</td>
<td>418296</td>
</tr>
<tr>
<td>1488000</td>
<td>108023963</td>
<td>245.25</td>
<td>0.70</td>
<td>224372</td>
<td>418332</td>
</tr>
</tbody>
</table>

Values for the case with packets of 40 bytes mean length and 1K analysis load in Ksensor (two-processor probe).

TABLE I.

After that, the Norton equivalent of the traffic monitoring system is obtained. As explained before, it is necessary to compute the expressions of the repeated topology several times. Since we validate the model for Ksensor, the values of the parameters related to the number of processors are $K_c=1$ and $K_f=K_a=2$. Regarding the probability $q_a$, every test has been configured with $q_a=1$. Table I shows the values of the Norton equivalent of the traffic monitoring system as intermediate results. As they are state-dependent service rates, they have two values, for $n=1$ and $n=2$.

Finally, applying the solution for the closed network model with incoming traffic, we get the system throughput and it can be plotted like in Fig. 9.
As mentioned, the model’s main result is the theoretical throughput and, as seen in Fig. 9, it can be compared to the throughput measured by the experimental tests (LAB in Fig. 9). As we can observe, with N=40, the results are nearly the same. This process, which has been illustrated for the case of 1K analysis load, has also been done for the other cases (null, 5K and 25K). Acceptable results are obtained too.

5 Conclusions and Future Work

In this paper we have presented and analytical model that represents multiprocessor traffic monitoring systems. The model is generic and quantifies the system performance.

Initially, we detect experimentally an apparently strange behavior of the parameter called softIRQ times per packet in the traffic monitoring system. This addresses us to set out a model based in a closed queuing network which considers the capturing, filtering and analysis stages as well as the possible packet losses while the packet goes from the network to the monitoring application. We obtain the model’s analytic solution, identifying a topology repeated at different levels of abstraction and applying the Norton equivalent to simplify the model. Then, the model is validated comparing theoretical results with experimental measurements over a prototype called Ksensor. In the validation process we make use of a testing architecture that not only measures the performance, it also provides values for some necessary input parameters of the mathematical model. As seen in the validation section, the obtained results are acceptable. Therefore, the model is able to represent precisely the behavior experimentally observed and also to evaluate the performance of the network traffic monitoring system, considering the most representative parameters like throughput, number of processors, analysis load and so on.

Moreover, the model is useful to interpret the variability of the processing time per packet in the capturing stage. The key is the parameter identified as p_QLQ in the model. In this work which combines the analytical study with experimental measurements, we have observed that this parameter is the most significant in the packet losses of the capturing stage. So, with high network rates, packet losses can be important and the smaller number of packets that reach the monitoring application require, in average, less CPU time.

Despite the fact that the conclusions have been satisfactory with regard to the behavior of the model, there are some aspects to be considered in the near future:

- We have presented a generic model for a traffic monitoring system, but we have only validated with the prototype Ksensor. We expect to adapt and validate the model to other platforms with more than two processors and over higher speed networks. We are especially interested in multicore systems and software probes under virtual machines.
- We have assumed Poisson processes and exponential service times and the results have been acceptable. However, we consider interesting to study the application of other distributions.

All in all, we believe this work is one step for a better understanding of co-locating capturing process and monitoring applications. We can understand how to best process packets and whether new modeling techniques can be applied to perform network measurement.

6 References