# Modeling and Analysis of Application Throughput in Gigabit Networks

K. Salah and K. El-Badawi
*Department of Information and Computer Science*
*King Fahd University of Petroleum and Minerals*
*Dhahran 31261, Saudi Arabia*
*Email: {salah,elbadawi}@kfupm.edu.sa*

**Abstract**

*This paper presents an analytical study of the impact of Gigabit network traffic on the overall user-perceived performance of network elements such as PC-based routers, servers, and workstations. Application processes can starve as CPU power of network elements is consumed by the overhead involved in handling interrupts and processing incoming packets. The potential of having a high network bandwidth has little value in practice if no CPU power is left for applications to process or forward data. In this paper, we present an analytical study of system and application throughputs and CPU utilization for hosts with interrupt-driven kernels. We develop analytical models based on queueing theory and Markov processes. We consider hosts with PIO and DMA design options. Our analysis work can be valuable for engineering and designing certain system parameters. It becomes imperative to understand how a host system and its applications would behave when subjected to low and very-high network loads. Simulations and reported experimental results show that our analytical models are valid and give an adequate approximation.*

**KEYWORDS**: Gigabit Networks, Operating Systems, Interrupts, Receive Livelock, Modeling and Analysis, Performance Evaluation.

## 1 Introduction

These days a massive deployment of 1-Gigabit and 10-Gigabit Ethernet network devices have taken place. With such large network bandwidth, the network packets arrive very close to each other, causing the kernel to spend most of its time processing the incoming packets. In Gigabit networks, the arrival rate of the incoming packet can exceed the host's processing rate. If no CPU power is left for the lower priority applications, the perceived performance by the user will significantly be degraded. This user-perceived performance can be reflected in a router that is not transmitting any packets or an interactive application that is not responding at all.

Interrupt-driven systems tend to perform very badly under such heavy network loads. Interrupt-level handling, by definition, has absolute priority over all other tasks. If interrupt rate is high enough, the system will spend all of its time responding to interrupts, and nothing else will be performed; and hence, the system throughput will drop to zero. This situation is called *receive livelock* [2]. In this situation, the system is not deadlocked, but it makes no progress on any of its tasks, causing any task scheduled at a lower priority to starve or not have a chance to run. At low packet arrival rates, the cost of interrupt overhead for handling incoming packets are low. However, interrupt overhead cost directly increases with an increase of packet arrival rate, causing *receive livelock.*

The receive livelock was established by experimental work on real systems in [2-4]. A number of solutions have been proposed in the literature [1,3,5-12] to address network and system overhead and improve the OS performance. Some of these solutions include interrupt coalescing, OS-bypass protocol, zero-copying, jumbo frames, polling, pushing some or all protocol processing to hardware, etc. In most cases, published performance results are based on research prototypes and experiments. However none of these solutions studied analytically the impact of interrupt overhead on OS performance.

In [16], a preliminary delay-throughput analysis was presented for interrupt-driven kernels when utilizing DMA in high-speed networks such as that of Gigabit Ethernet. In this paper, we present analytical models that are based on queueing theory and Markov processes. We consider and study two design options. One is for PIO-based design in which network adapters are not equipped with DMA engines, and the other is for DMA-based design in which network adapters are equipped with DMA engines. As opposed to prototyping and simulation, these models can be utilized to give a quick and easy way of studying the host performance in high-speed and Gigabit networks. These models yield insight into understanding and predicting the performance and behavior of interrupt-driven systems and its applications at low and at very-high network traffic.

In this paper, we study the performance in terms of system and application throughputs and CPU utilization. CPU utilization is an important evaluation criterion to study. The potential of high network bandwidth has no value in practice if not much CPU power is left to process data at the application level. A good design must ensure that the user-mode applications are left with sufficient CPU power to send and absorb network traffic. Our analysis is based on queueing theory and can be easily extended to study mean latency, mean waiting time, mean number of packets in system and queues, blocking probability, etc. In addition, our analytical work can be important for engineering and designing various NIC and system parameters. These parameters may include the proper service times for ISR handling and protocol processing, buffer sizes, CPU bandwidth allocation for protocol process and application, etc.

The rest of the paper is organized as follows. Section 2 presents models and analytical study to examine the impact of heavy network traffic on the host performance. Section 4 describes a discrete-even simulation to verify analysis. Section 3 gives numerical results and examples for analysis, simulation, and real experiments.. Finally, Section 5 concludes the study and identifies future work.

## 2 Analysis

In this section we present an analytical study to examine the impact of interrupt overhead and OS network protocol processing on kernel and application throughputs. First we define the system parameters. Let $\lambda$ be the mean incoming packet arrival rate, $\mu_s$ be the mean protocol processing rate carried out by the kernel, and $\mu_a$ be the mean processing rate carried out by the user-mode application process. Note that $1/\mu_s$ is the average time the system takes to process the incoming packet and deliver it to the user application. This time includes primarily the network protocol stack processing carried out by the kernel, excluding any time disruption due to interrupt handling. Also, $1/\mu_a$ is the average time to process one packet by the user application, excluding any time disruption due to interrupt handling or protocol processing. Let $1/r$ be the mean interrupt handling time, which is basically the interrupt service routine time for handling incoming packets. $1/r$ includes the interrupt-context switching overhead, servicing the packet which may include copying the packet from NIC to kernel memory, and lastly notifying the kernel to start the protocol processing for the received packet.

After the notification of the arrival of a new packet, the kernel will process the packet by first examining the type of frame being received and then invoking immediately the proper handling stack function or protocol, e.g. ARP, IP, TCP, etc. The packet will remain in the kernel or system memory until it is discarded or delivered to the user application. The network protocol processing for packets carried out by the kernel will continue as long as there are packets available in the system memory buffer. However, this protocol processing of packets can be interrupted by ISR executions as a result of new packet arrivals. This is so because packet processing by the kernel runs at a lower priority than the ISR.

There are two possible system delivery options of packet to the user applications. The first option is to perform an extra copy of packet from kernel space to user space. This is done as part of the OS protection and isolation of user space and kernel space. This option will stretch the time of protocol processing for each incoming packet. A second option eliminates this extra copy using different techniques described in [6,7,8,13,14,15]. The kernel is written such that the packet is delivered to the application using pointer manipulations. Our

analytical model captures both options. The only difference is in the protocol processing time. The second option will have a smaller processing time than the first.

Throughout our analysis, we assume the following:

i) It is reasonable not to assume the times for application processing, protocol processing, or ISR handling to be constant. These times change due to various OS activities. For example ISR handling for incoming packets can be interrupted by other interrupts of higher priority, e.g. timer interrupts. Also, protocol and application processing can be interrupted by higher priority kernel tasks, e.g. scheduler. For our analysis, we assume these service times to be exponential. In Section 4, we demonstrate that this assumption gives an adequate approximation.

ii) The network traffic follows a Poisson process, i.e. the packet interarrival times are exponentially distributed.

iii) The packet sizes are fixed. This assumption is true for Constant Bit Rate (CBR) traffic such as uncompressed interactive audio and video conferencing. This assumption is also true for all ATM traffic in which cells of a fixed size are always used.

### 2.1 Limitations

Our analytical models assume the packet arrivals are Poisson, and the packets are of a fixed size. In practice, network packets are not always fixed in size, and their arrivals do not always follow a Poisson process. An analytical solution becomes intractable when considering variable-size packets and non-Poisson arrivals. As we will demonstrate in Section 4, it turns out that our model with the above assumptions gives a good approximation to real experimental measurements. The impact of having a constant network traffic instead of a Poisson is studied using simulation in this paper and results are shown and compared to those of Poisson. However, having variable-size packets, e.g. Jumbo frames, and other traffic distributions, e.g. bursty traffic [17], are currently being studied by the authors using simulations and results are expected to be reported in the near future.

### 2.2 The Model

Based on the assumptions above, the system can be modeled as an open tandem queueing network consisted of two M/G/1/B queues in series, as shown in Figure 1. Using Jackson's result, we can analyze each queueing system separately as two independent M/G/1/B queues. We use M/G/1/B queues as opposed to M/G/1 since the arrival rate can go beyond the service rate, i.e., $\rho > 1$. This assumption is a must for Gigabit environment where under heavy load $\lambda$ can be very high compared to $\mu$. In this model, we assume only one user-mode

application running on the host. This model can be adequate for PC-based network elements such as proxies, firewalls, gateways, or routers. For these network elements, the application queue represents the primary application/function of these elements such as logging, monitoring, filtering, address or port translation, caching, forwarding, etc. For servers and workstations running multiple applications, the model can be extended using queueing theory techniques, with each application having its own processing rate and its own probabilistic Poisson arrival.

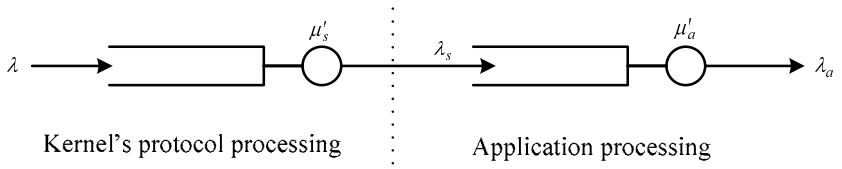


**Figure 1. A model of two M/G/1/B queues in series**

### 2.2.1 Protocol Processing Queue

One may think the protocol processing queue with interrupt disruption can be simply modeled as a priority queue with preemption in which there are two arrivals of different priorities. The first arrival is the arrival of ISRs, and has the higher priority. The second arrival is the arrival of incoming packets, and has the lower priority. As noted the ISR execution preempts protocol processing. However this is an invalid model because ISR handling is not counted for every packet arrival. ISR handling is ignored if the system is servicing another interrupt of the same level. In other words, if the system is currently executing another ISR, the new ISR which is of the same priority interrupt level will be masked off and there will be no service for it. We use instead a queueing model based on the mean effective service rate $\mu'_s$ , expressed in terms of CPU percentage availability for protocol processing.

$\mu'_s$ = Rate at which packets get processed by the kernel's network protocol given that the CPU is available for protocol processing, i.e. the CPU is not handling ISR. Let us denote $p_{0,0}$ to be the percentage of CPU availability for protocol processing. Hence,

$$\mu'_s = \mu_s \cdot p_{0,0}, \quad (1)$$

Since $\mu'_s$ is exponential, the protocol processing queue is an M/M/1/B with a mean kernel or system throughput expressed as

$$\lambda_s = \mu'_s \cdot (1 - p_0), \quad (2)$$

where $p_0$ is the probability of zero packets being processed by the kernel's network protocol stack. With $\rho = \lambda_s / \mu'_s$ , $p_0$ is given by

$$p_0 = \begin{cases} \dfrac{1-\rho}{1-\rho^{B+1}} & (\rho \neq 1), \\ \dfrac{1}{B+1} & (\rho = 1). \end{cases} \tag{3}$$

### 2.2.2 Application Processing Queue

Similarly, the mean effective service rate for the user application can be expressed in terms of the available CPU power left for the user application. The CPU availability percentage for the user application can be determined using two approaches which give the same outcome. One approach, the CPU percentage available for other processing is the probability when there is no ISR handling and there are no packets being processed by the protocol stack. This can be expressed as $p_{0,0} \cdot p_0$. A second approach, the CPU availability percentage for other processing is 1 – {The sum of the CPU utilization for ISR handling and the CPU utilization for protocol processing}. This can be expressed as $1-\{(1-p_{0,0})+(1-p_0)\cdot p_{0,0}\}$, or simplified further to $p_{0,0} \cdot p_0$. Therefore,

$$\mu'_a = \mu_a \cdot p_{0,0} \cdot p_0, \tag{4}$$

where $p_0$ is given by equation (3) with $\rho = \lambda_s / \mu'_s$ .

It is observed that from equation (4) that $\mu'_a$ is exponential, and from equation (2) that $\lambda_s$ is Poisson. Therefore the application processing queue can be modeled as an M/M/1/B with a mean application throughput expressed as

$$\lambda_a = \mu'_a \cdot (1 - p_0), \tag{5}$$

where $p_0$ is the probability of zero packets being processed by the application. $p_0$ is given by equation (3) with $\rho = \lambda_a / \mu'_a$ .

## 2.3 Available CPU Bandwidth for Protocol Processing

In this section, we determine the available CPU bandwidth for protocol processing, i.e. $p_{0,0}$. Knowing $p_{0,0}$, we can find the mean effective service rates for kernel's protocol processing and user application of equation (1) and equation (4), respectively. We consider two design options: PIO and DMA.

### 2.3.1 PIO-Based Design

Traditional network adapters or Network Interface Cards (NICs) as those of 10Mbps Ethernet and 16Mbps Token Ring are not equipped with DMA engines. The copying of an arrived packet from NIC buffer to host kernel memory is performed by the CPU as part of ISR handling for each incoming packet. This method of copying is known as PIO (Programmed I/O). In PIO, the CPU during the ISR handling sits in a tight loop copying data from the NIC memory into the host memory. After the packet is copied, the ISR then sets a software interrupt to trigger packet protocol processing. It is very possible that one or more incoming packets arrive during the execution of the ISR. For this, the ISR handling must not exit unless all incoming packets are copied from the NIC to the kernel memory. As more packets arrive during ISR handling, i.e. the arrival rate $\lambda$ becomes high, most of the CPU power will be consumed by ISR handling. And therefore, the rate of protocol processing for incoming packets carried out by the kernel will be degraded to zero.

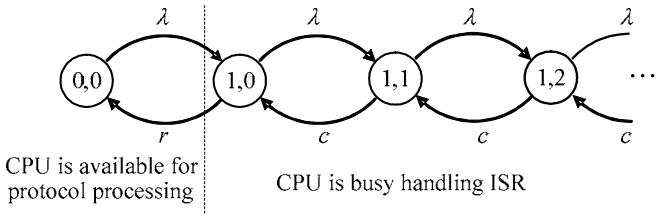


**Figure 2. Markov state transition diagram for modeling CPU usage with PIO**

In order to determine the CPU availability percentage for protocol processing, $p_{0,0}$, we use a Markov process to model the CPU usage, as illustrated in Figure 2. We assume the copying of the packet from NIC to kernel memory is exponentially distributed with a mean service time of $1/c$. As shown in the figure, the Markov process has state (0,0) and states (1,*n*). State (0,0) represents the state where the CPU is available for protocol processing. States (1,*n*) with $0 < n < \infty$ represents the state where the CPU is busy handling interrupts. *n* denotes the number of interrupts that are batched or masked off during ISR handling. Note that *n*+1 denotes the number of packet arrivals during ISR handling. In other words, state (1,0) means there are no interrupts being masked off and the CPU is busy handling an ISR with one packet arrival. State (1,1) means that one interrupt has been masked off and the CPU is busy handling an ISR with two packet arrivals: one packet will be serviced with a mean rate of *r* and the other with a mean rate of *c*.

The steady-state solution of the Markov chain, shown in Figure 2, can be expressed as

$$p_{1,n} = \frac{\lambda^{n+1}}{r\,c^{n}}\,p_{0,0} \qquad (n = 0,1,2,\cdots). \tag{6}$$

Using the boundary condition that $p_{0,0} + \sum_{n=0}^{\infty} p_{1,n} = 1$, we get

$$p_{0,0} + \sum_{n=0}^{\infty} \frac{\lambda^{n+1}}{r\, c^n} p_{0,0} = 1.$$

Hence,

$$p_{0,0} = \left[ 1 + \frac{\lambda}{r} \sum_{n=0}^{\infty} \left( \frac{\lambda}{c} \right)^n \right]^{-1} = \frac{rc - r\lambda}{rc - r\lambda + c\lambda}. \tag{7}$$

The geometric series $\sum_{n=0}^{\infty} (\lambda / c)^n$ converges only for $\lambda < c$. For $\lambda > c$, the geometric series does not converge, i.e., goes to infinity, and $p_{0,0}$ becomes zero. Thus the CPU availability percentage for protocol processing ( $p_{0,0}$ ) can be expressed as

$$p_{0,0} = \begin{cases} \dfrac{rc - r\lambda}{rc - r\lambda + c\lambda}, & \lambda < c \\ 0. & \lambda \geq c \end{cases} \tag{8}$$

### 2.3.2 DMA-Based Design

PIO-based design can be an attractive option when considering factors such as cost, simplicity, and speed and efficiency in copying relatively small-size packets [18]. However, a major drawback for a PIO-based design is burdening the CPU with copying incoming packets from the NIC to kernel memory. In order to save CPU cycles consumed in copying packets, major network vendors equip high-speed NICs with DMA engines. These vendors include 3Com, HP, Alteon owned now by Nortel, Sundace, and NetGear. NICs are equipped with a receive Rx DMA engine and a transmit Tx DMA engine. A Rx DMA engine handles transparently the movement of packets from the NIC internal buffer to the host system memory. A Tx DMA engine handles transparently the movement of packets from the host memory to the NIC internal buffer. Both DMA engines operate in a bus-master fashion, i.e. the engines request access to the PCI bus instead of waiting to be polled. It is worth noting that the transfer rate of incoming traffic into the kernel memory across the PCI bus is not limited by the throughput of the DMA channel. These days a typical DMA engine can sustain over 1 Gbps of throughput across the bus [19].

It is important to note that the device driver for the network adapters is typically configured such that an interrupt is generated after the incoming packet has been completely DMA'd into the host system memory. In order to minimize the time for ISR execution, ISR handling mainly sets a software interrupt to trigger the protocol processing for the incoming packet. Please note in this situation if two or more packets arrive during

an ISR handling, the ISR time for servicing all of these packets will be the ISR time for servicing a single packet, with no extra time introduced.

In order to find the CPU availability percentage for protocol processing, $p_{0,0}$, we use a Markov process to model the CPU usage, as illustrated in Figure 3. The process has state (0,0) and states (1,*n*). State (0,0) represents the state where the CPU is available for protocol processing. States (1,*n*) with $0 < n < \infty$ represents the state where the CPU is busy handling interrupts. *n* denotes the number of interrupts that are batched or masked off during ISR handling. Note that *n*+1 denotes the number of packet arrivals during ISR handling. Therefore, state (1,0) means there are no interrupts being masked off and the CPU is busy handling an ISR with one packet arrival. State (1,1) means that one interrupt has been masked off and the CPU is busy handling an ISR with two packet arrivals. Both of these packets will be serviced together with a mean rate *r* of servicing only one packet.

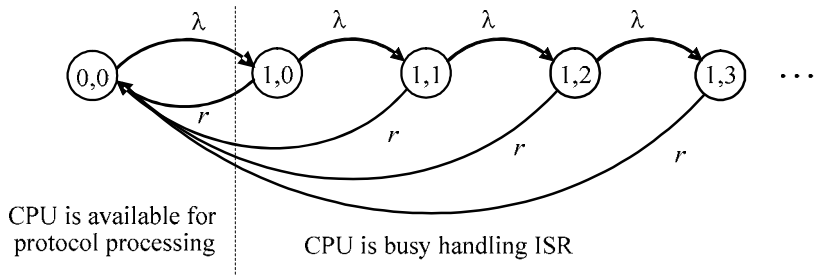


**Figure 3. Markov state transition diagram for modeling CPU usage with DMA**

The steady-state difference equations can be derived from $\mathbf{0} = \boldsymbol{pQ}$, where $\boldsymbol{p} = \{p_{0,0}, p_{1,0}, p_{1,1}, p_{1,2}, \cdots\}$ and $\boldsymbol{Q}$ is the rate-transition matrix and is defined as follows:

$$\boldsymbol{Q} = \begin{bmatrix} -\lambda & \lambda & 0 & 0 & 0 & \cdots \\ r & -(\lambda + r) & \lambda & 0 & 0 & \cdots \\ r & 0 & -(\lambda + r) & \lambda & 0 & \cdots \\ r & 0 & 0 & -(\lambda + r) & \lambda & \cdots \\ r & 0 & 0 & 0 & -(\lambda + r) & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \end{bmatrix}$$

This will yield $-\lambda p_{0,0} + r(p_{1,0} + p_{1,1} + p_{1,2} + \cdots) = 0.$

Since we know that $p_{0,0} + \sum_{n=0}^{\infty} p_{1,n} = 1$, then

$$-\lambda p_{0,0} + r(1 - p_{0,0}) = 0.$$

Solving for $p_{0,0}$, we thus have

$$p_{0,0} = \frac{r}{\lambda + r}, \tag{9}$$

and $1 - p_{0,0} = \frac{\lambda}{\lambda + r}$. Therefore, the CPU availability percentage for protocol processing and the CPU utilization for handling interrupts are $r/(\lambda + r)$ and $\lambda/(\lambda + r)$, respectively.

### 2.4 Kernel Livelock

A critical operating point for the kernel's protocol processing of the host is the condition at which the system throughput becomes zero. This condition is where the host's kernel would livelock. By examining equation (1) and equation (2), system throughput $\lambda_s$ can be zero if there is no packet being processed by the kernel or the mean effective protocol processing rate $\mu_s'$ is zero. $\mu_s'$ reaches zero when $p_{0,0}$ becomes zero.

Therefore for PIO, the kernel would livelock when $\lambda \geq c$. See equation (8). It is important to note that the receive livelock always occurs at the same point regardless of improving the network protocol processing for packets. In another words, utilizing a mechanism such as zero-copy, in which the protocol processing is reduced, does not eliminate receive livelock. For DMA, the kernel will not livelock unless $\lambda \to \infty$. See equation (9). As opposed to PIO, the receive-livelock point for DMA occurs when $\lambda \to \infty$.

### 2.5 Application Livelock

Another interesting operating point is the condition at which the available CPU bandwidth for user applications becomes zero. This is where the user applications would starve, and the total CPU utilization for ISR handling and protocol processing reaches 100%. This is also the saturation point where the host can not keep up with the offered network load. This point is also known as the "cliff" point of system throughput. It is where the throughput starts falling as the network load increases. This point precisely occurs when $\lambda = \mu_s'$. For PIO, this saturation point can be derived and expressed as

$$\lambda = \frac{\sqrt{r^2(c-\mu_s)^2 + 4rc^2\mu_s} - r(c+\mu_s)}{2(c-r)}, \tag{10}$$

and for DMA, we get

$$\lambda = \frac{r}{2}\left(\sqrt{1 + 4\frac{\mu_s}{r}} - 1\right). \tag{11}$$

## 3 Simulation

In order to verify the analytical models, a discrete-event simulation was developed and written in C language. The simulation follows closely and carefully the guidelines given in [20]. Figures 4 and 5 show the simulation flowchart for PIO and DMA, respectively. Table 1 describes the events. There are five events for PIO, and four for DMA. Except for the ARRIVAL event, each event has a time, status, and priority. An event status can be IDLE, BUSY, or SUSPEDNDED. IDLE indicates the event has not been selected by the scheduler, i.e, not being served by the CPU. BUSY indicates the event is being served by the CPU. SUSPENDED indicates the event was BUSY but got preempted by a higher priority event. Only IP_DEPART and APP_DEPART events can have SUSPENDED status. The selection of the next event by the scheduler is based on the event's time, status, and priority. Whenever a SUSPENDED event is selected again to run (i.e., resumed running), its finish time will incur the service times of all higher priority events which occurred between its suspension and its resumption. The simulation has three buffers implemented as FIFO queues: NIC, IP, and application. These queues hold the value of the arrival time of each packet. The simulation run ends when the total number of events reaches five millions.

**Table 1. Simulation events**

| Event | Description |
|---|---|
| ARRIVAL | Occurs when a new packet arrives to NIC. |
| ISR | Occurs when a packet received successfully by the NIC and the CPU is not busy handling an ISR. |
| ISR_COPY | Used only in PIO during ISR handling. CPU copies the incoming packet from NIC to IP buffer. This event occurs due to an ISR event, or due to having more packets in the NIC buffer. |
| IP_DEPART | Occurs whenever NIC buffer is empty and IP buffer has packets. IP_DEPART indicates the completion of IP processing of one packet. This processing includes copying the packet from IP buffer to application buffer. |
| APP_DEPART | Occurs whenever IP buffer is empty and application buffer has packets. APP_DEPART indicates the completion of handling a packet by the user application. |

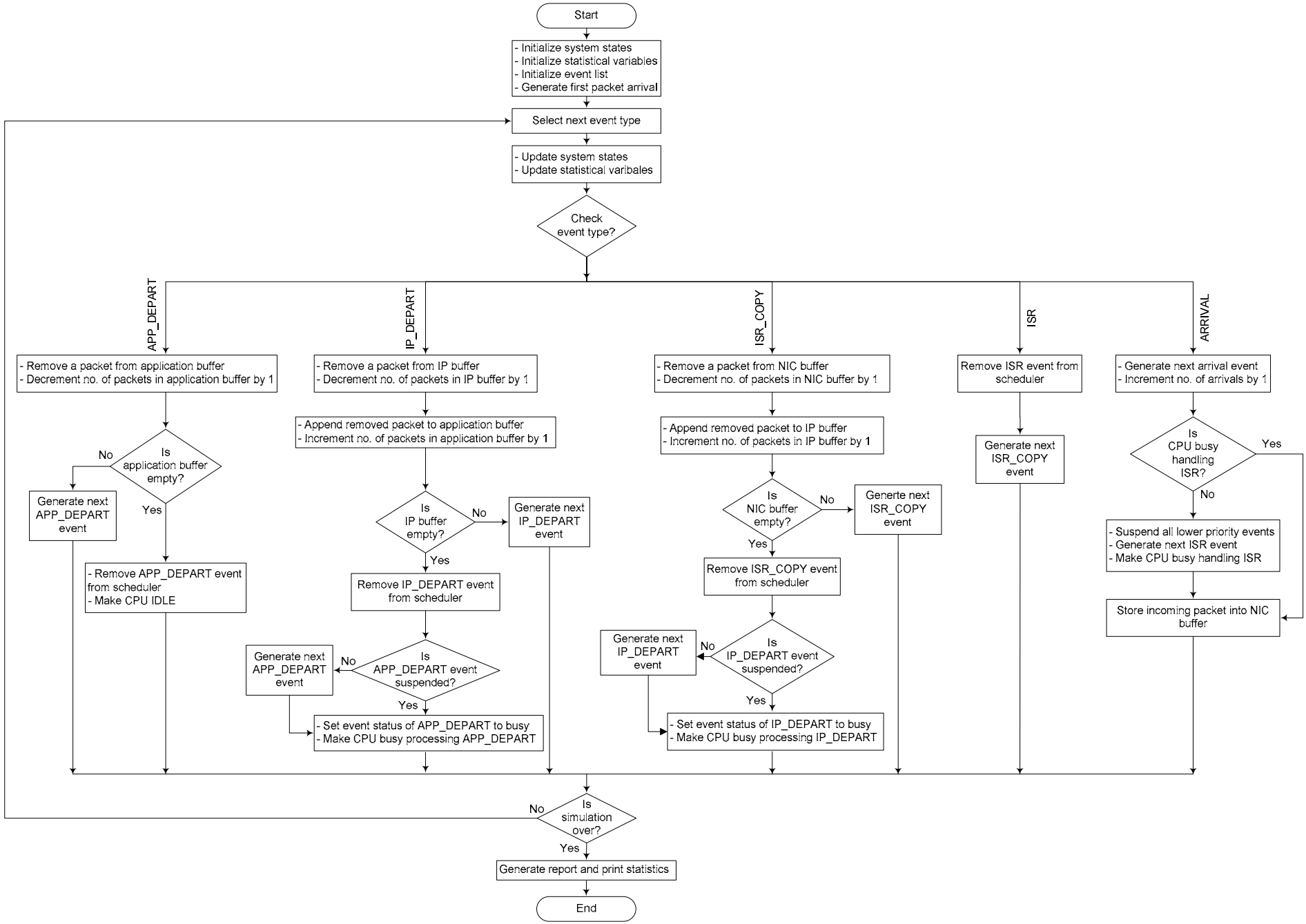


**Figure 4. Simulation flowchart for PIO**

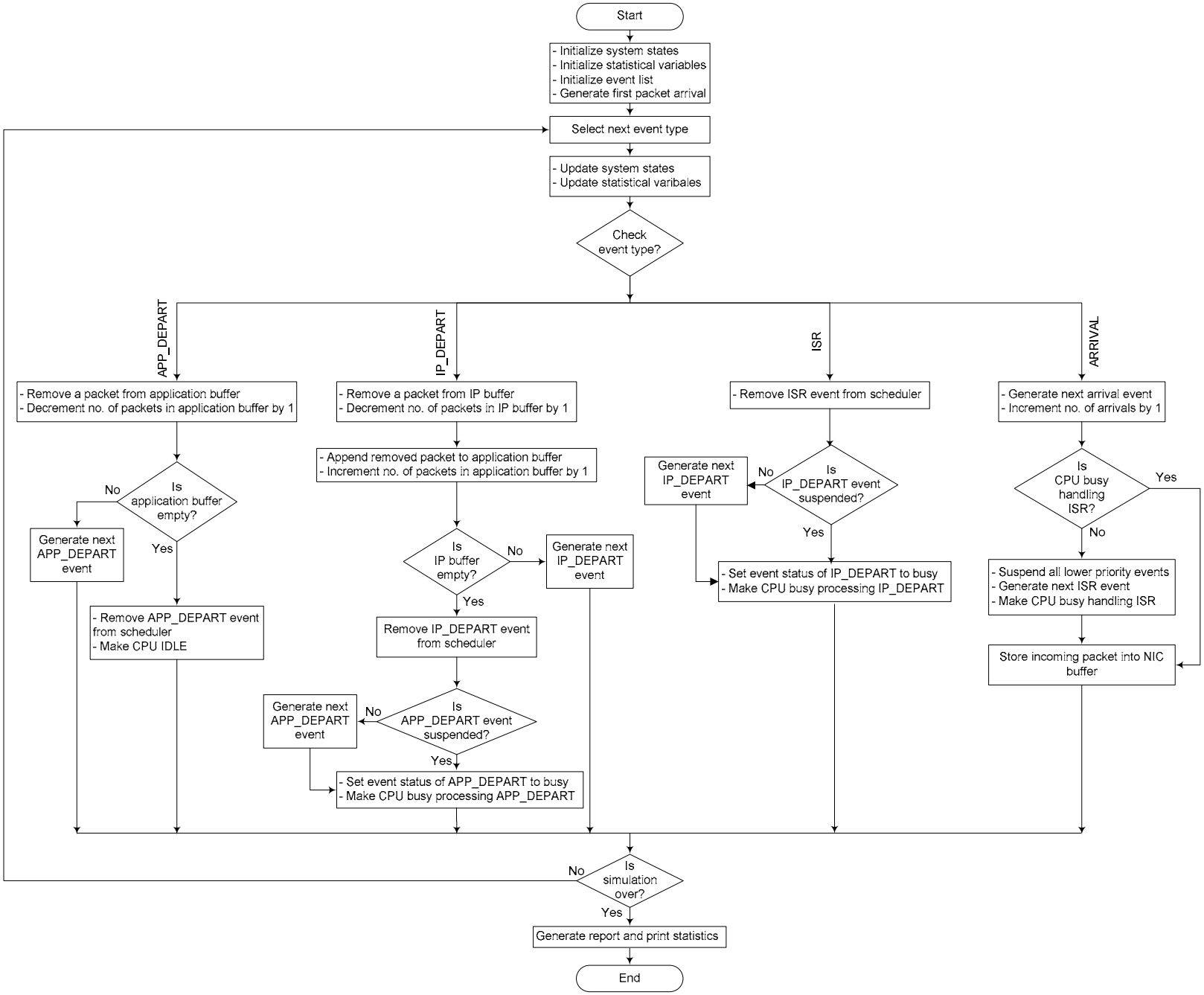


**Figure 5. Simulation flowchart for DMA**

## 4 Numerical Examples

In this section, we report and compare results of system and application throughputs and CPU utilization for hosts with PIO and DMA design options. Numerical examples are given for both analysis and simulation results. For validation, we compare our analysis of system and application throughputs to the PIO experimental results reported in [3] and the DMA experimental results reported in [4].

In [3], the experiment basically consisted of a target system of a router, DECstation 3000/300 Alpha-based, running Digital UNIX V3.2 OS with 10Mbps Ethernet NICs with no DMA. A traffic of fixed-size packets was generated back-to-back to the router using an infinite loop running at the application level on another host. As measured by [3], the mean service time for ISR ($1/r$) was 95 μ seconds which included the copying the packet from NIC to kernel memory. This mean copy time from NIC to kernel memory ($1/c$) was 55 μ seconds. The mean protocol processing time ($1/\mu_s$), which included copying of packet data to user space, was 150 μ seconds. The mean application processing rate ($\mu_a$) was 7.7 Kpps.

In [4], the experiment basically consisted of a PC-based router, 450 MHz Pentium III, running Linux 2.2.10 OS with two Fast-Ethernet NICs with DMA. Similarly, a traffic of fixed-size packets was generated back-to-back to the router. As measured by [4], the mean service time for ISR ($1/r$) was 7.7 μ seconds and the mean protocol processing time ($1/\mu_s$) was 9.7 μ seconds. This time included processing the packet as well as forwarding it. In [4], the Linux kernel was modified such that both packet processing and forwarding were performed at the kernel level as one task. In [4], there was no measurement for the mean application time ($1/\mu_a$). For our numerical examples, we assume ($1/\mu_a$) to be 15 μ seconds.

For all analysis and simulation runs, we fix $B$ to a size of 1000. Both Figure 5 and Figure 6 show system and application throughputs and CPU utilization with PIO and DMA, respectively. The discrete-event simulation results are very much in line with those of analysis, and not shown in the figures for clarity. As for validation, we compare the experimental results of to those of analysis. CPU utilization and availability using real experiments were not measured in [4]. Only CPU availability for user-mode processes was measured in [3] and its results are shown in Figure 5. From the figures, it is depicted that the analysis results give an adequate approximation to real experimental measurements. For throughputs, both figures depict that the analysis results match exactly those of experimental at light load. At high load, there is a slight difference for both PIO and DMA. This difference can be contributed to the arrival characteristics of incoming packets. Our analysis assumed a Poisson arrival; however, as stated earlier, the inter-arrival times of the packets are not purely

exponential. In the experiments, that the packets were generated back-to-back by another host with almost a constant rate. Another factor that may contribute to the difference of experimental and analysis results for throughputs and CPU availability can be linked to the fact that the distribution of the service times for protocol processing and ISR handling are not exponential. In our analysis, we assumed exponential distribution for both of the theses service times. Other factors can also be related to internal caching, system background processes, user-to-kernel mode switching, and measurement overhead.

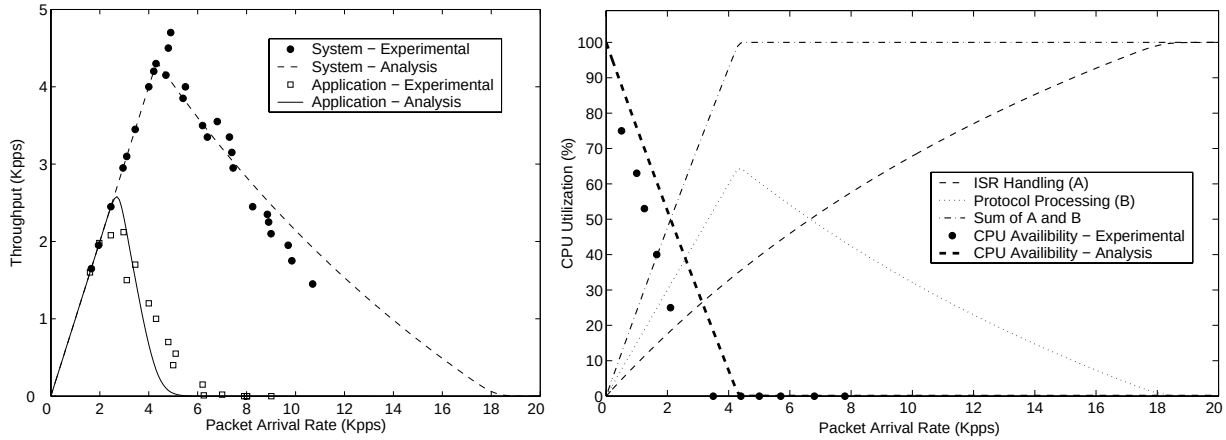


**Figure 5. Throughput and CPU utilization with PIO**

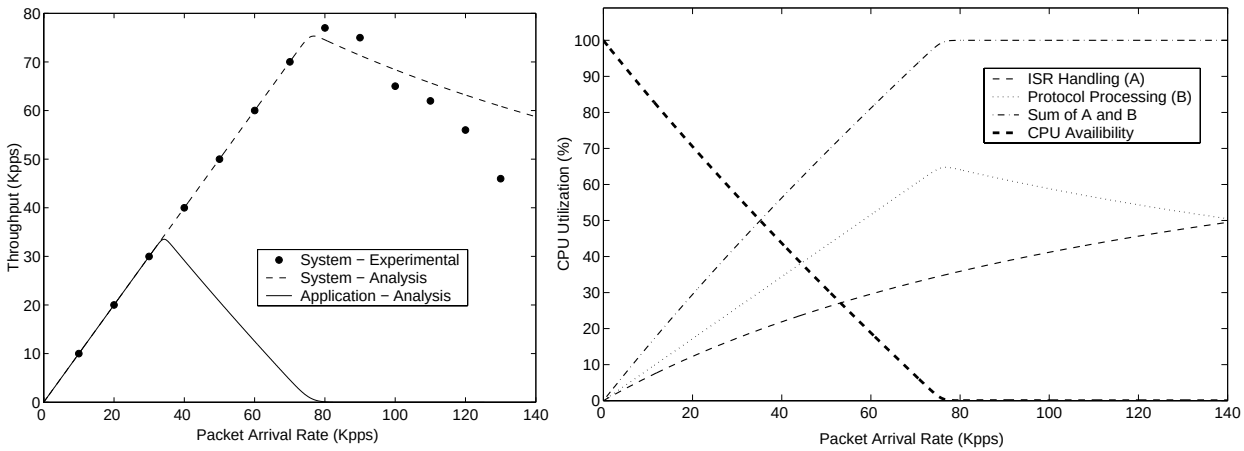


**Figure 6. Throughput and CPU utilization with DMA**

When comparing PIO and DMA throughputs, one can conclude that DMA system throughput does not fall rapidly to zero, but it gradually decreases as the system load of packet arrivals increases. It can be noted from Figure 5 that the kernel livelock point for PIO occurs at $\lambda = c = 18{,}182$ pps. At this point the corresponding CPU utilization for protocol processing is zero, and the CPU utilization for ISR handling is at 100%. On the other hand for DMA, the receive-livelock point is not shown within the scope of the network offered load.

Remember that for DMA, the receive-livelock point does not occur unless $\lambda \to \infty$. As for the application livelock point, it is depicted from both figures that user applications would have zero throughput at an arrival rates of 4,328 pps and 67,750 pps, for PIO and DMA, respectively.

## 5 Conclusion

We developed analytical models to study the impact of interrupt overhead caused by Gigabit network traffic on user-perceived performance. We studied the performance in terms of system and application throughputs as well as CPU utilization for hosts with PIO and DMA when subjected to light and heavy network loads. As noted, degraded throughputs and no CPU power for applications can be encountered due to heavy network loads. If the system is poorly designed, these penalties are quite high and can hurt the overall performance. Our analysis effort provided equations that can be used to easily and quickly predict the system performance and behavior when engineering and designing network adapters, device drivers, and OS network and communication software. Given a worst-case network load, acceptable performance levels for throughputs and CPU availability can be reached by choosing between DMA vs. PIO design options and by finding the proper system parameters for protocol processing and ISR times. An acceptable performance level varies from one system requirement to another and depends on the worst tolerable throughputs and CPU availability. Our analytical models were verified by simulation. Also reported experimental results show that our analytical models give a good approximation. As demonstrated in the paper, receive livelock phenomenon is far worse in PIO than DMA. For DMA, the receive livelock only occurs when the packet arrival rate reaches infinity. Therefore, utilizing DMA engines for Gigabit network adapters becomes very important design and implementation consideration in order to minimize the negative impact of interrupt overhead on system performance. The impact of generating variable-size packets instead of fixed-size and bursty traffic instead of Poisson is being studied by the authors using simulation, and results are expected to be reported in the near future. A lab experiment of 1-Gigabit links is also being set up to measure and compare the performance of different system metrics. As a further work, we will study and evaluate the performance of the different proposed solutions for minimizing and eliminating the interrupt overhead caused by heavy network loads.

## References


[1] I. Kim, J. Moon, and H. Y. Yeom, "Timer-Based Interrupt Mitigation for High Performance Packet Processing, " Proceedings of 5th International Conference on High-Performance Computing in the Asia-Pacific Region, Gold Coast, Australia, September 2001.

[2] K. Ramakrishnan, "Performance Consideration in Designing Network Interfaces," *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 2, February 1993, pp. 203-219.

[3] J. Mogul, and K. Ramakrishnan, “Eliminating Receive Livelock In An Interrupt-Driven Kernel,” *ACM Trans. Computer Systems,* vol. 15, no. 3, August 1997, pp. 217-252.

[4] R. Morris, E. Kohler, J. Jannotti, and M. Kaashoek, “The Click Modular Router,” *ACM Transactions on Computer Systems*, vol. 8, no. 3, August 2000, pp. 263-297.

[5] A. Indiresan, A. Mehra, and K. G. Shin, “Receive Livelock Elimination via Intelligent Interface Backoff,” TCL Technical Report, University of Michigan, 1998.

[6] P. Druschel, “Operating System Support for High-Speed Communication,” *Communications of the ACM*, vol. 39, no. 9, September 1996, pp. 41-51.

[7] P. Druschel, and G. Banga, “Lazy Receive Processing (LRP): A Network Subsystem Architecture for Server Systems,” Proceedings Second USENIX Symposium. on Operating Systems Design and Implementation, October 1996, pp. 261-276.

[8] P. Shivan, P. Wyckoff, and D. Panda, “EMP: Zero-copy OS-bypass NIC-driven Gigabit Ethernet Message Passing,” Proceedings of SC2001, Denver, Colorado, USA, November 2001.

[9] C. Dovrolis, B. Thayer, and P. Ramanathan, “HIP: Hybrid Interrupt-Polling for the Network Interface,” *ACM Operating Systems Reviews*, vol. 35, October 2001, pp. 50-60.

[10] Alteon WebSystems Inc., “Jumbo Frames,” http://www.alteonwebsystems.com/products/white_papers/jumbo.htm

[11] A. Gallatin, J. Chase, and K. Yocum, "Trapeze/IP: TCP/IP at Near-Gigabit Speeds", Annual USENIX Technical Conference, Monterey, Canada, June 1999.

[12] C. Traw, and J. Smith, "Hardware/software Organization of a High Performance ATM Host Interface," *IEEE JSAC*, vol.11, no. 2, February 1993.

[13] J. Brustoloni and P. Steenkiste, "Effects of Buffering Semantics on I/O Performance ," Proceedings Second USENIX Symposium. on Operating Systems Design and Implementation, October 1996, pp. 277-291.

[14] Z. Ditta, G. Parulkar, and J. Cox, "The APIC Approach to High Performance Network Interface Design: Protected DMA and Other Techniques," Proceeding of IEEE INFOCOM 1997, Kobe, Japan, April 1997, pp. 179-187.

[15] H. Keng and J. Chu, "Zero-copy TCP in Solraris," Proceedings of the USENIX 1996 Annual Technical Conference, January 1996.

[16] K. Salah and K. El-Badawi, "Performance Evaluation of Interrupt-Driven Kernels in Gigabit Networks", IEEE Globecom 2003, San Francisco, USA, December 1-5, 2003, pp. 3953-3957.

[17] W. Leland, M. Taqqu, W. Willinger, D. Wilson, "On the Self-Similar Nature of Ethernet Traffic", *IEEE/ACM Transaction on Networking*, vol. 2, pp. 1-15, 1994.

[18] P. Steenkiste, "A Systematic Approach to Host Interface Design for High-Speed Networks," *IEEE Computer magazine*, Vol. 24, No. 3, March 1994, pp. 44-52.

[19] 3Com Corporation, "Gigabit Server Network Interface Cards 7100xx Family," http://www.costcentral.com/pdf/DS/3COMBC/DS3COMBC109285.PDF

[20] A. Law and W. Kelton, *Simulation Modeling and Analysis*, McGraw-Hill, 2nd Edition, 1991.