Throughput-Delay Analysis of Interrupt-Driven Kernels with DMA Enabled and Disabled in High-Speed 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

Interrupt processing can be a major bottleneck in the end-to-end performance of high-speed networks. The performance of Gigabit network end hosts or servers can be severely degraded due to interrupt overhead caused by heavy incoming traffic. Under heavy network traffic, the system performance will be negatively affected due to interrupt overhead caused by the incoming traffic. In particular, excessive latency and significant degradation in system throughput can be experienced. In this paper, we present a throughput-delay analysis of such behavior. We develop analytical models based on queueing theory and Markov processes. In our analysis, we consider and model three systems: ideal, PIO, and DMA. In ideal system, the interrupt overhead is ignored. In PIO, DMA is disabled and copying of incoming packets is performed by the CPU. In DMA, copying of incoming packet is performed by DMA engines. For high-speed network hosts, both PIO and DMA can be desirable configuration options. The analysis yields insight into understanding and predicting the impact of system and network choices on the performance of interrupt-driven systems when subjected to light and heavy network loads. Simulations and reported experimental results show that our analytical models are valid and give a good approximation.

KEYWORDS: High-Speed Networks, Operating Systems, Interrupts, Receive Livelock, Modeling and Analysis, Performance Evaluation.

1. Introduction

Nowadays, a massive deployment of 1-Gigabit and 10-Gigabit for Ethernet network devices is taking place. However, existing operating systems still use for the most part the same mechanisms to handle both network processing and traditional I/O devices. Traditional operating systems were designed to handle network devices that interrupt on average rate of around 1000 packets per second, as is the case for shared 10Mbps Ethernet [1]. The cost of handling interrupts in these traditional systems was low enough that any normal system would spend only a fraction of its CPU time handling interrupts. The shift to higher packet arrival rate can subject a host to congestive collapse. Such a collapse is driven primarily by interrupt handling and overhead.

[**] Corresponding Author: Prof. K. Salah, PO Box 5066, ICS Department, KFUPM, Dhahran 31261, Saudi Arabia

Under heavy traffic load such as that of Gigabit networks, the performance of interrupt-driven systems can be degraded significantly, and thus resulting in a poor host performance perceived by the user. For one thing, every hardware interrupt, for every incoming packet, is associated with context switching of saving and restoring processor's state as well as potential cache/TLB pollution. More importantly, 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.

The receive livelock was established by experimental work on real systems in [2-4]. A number of solutions have been proposed in the literature [1,3,5-13] to address network and system overhead and improve the OS (Operating System) 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 little or no research has been done to study analytically the impact of interrupt overhead on OS performance. In [9,13], a simple calculation of the interrupt overhead was presented. In [9], a mathematical equation was given directly for application throughput based on packet length and cost of interrupt overhead per byte and per packet. In [13], the interrupt overhead was computed based on the arrival rate, ISR (Interrupt Service Routine) time, and a fixed cost of interrupt overhead. Both of these calculations are very simple. The calculations fail to consider complex cases such as interrupt masking, CPU utilization, and effect of ISR and its overhead on packet processing at OS and application levels. Moreover, the calculations fail to capture the receive livelock phenomenon and identify the saturation point of the host.

In [17], a preliminary delay-throughput analysis was presented for interrupt-driven kernels when utilizing DMA (Direct Memory Access) in high-speed networks such as that of Gigabit Ethernet. In this paper, we present three analytical models that are based on queueing theory and Markov processes. We first present an analytical model for the ideal system when interrupt overhead is ignored. We then present two models which describe the impact of high interrupt rate on system performance. One model is for PIO (Programmable Input/Output) option in which DMA is disabled or for host in which the network adapters are not equipped with DMA engines. The other model is for DMA option 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 receive livelock phenomenon and system performance in high-speed and Gigabit networks. These models yield insight into understanding and predicting the performance and behavior of interrupt-driven systems at low and at very-high network traffic. The models can also be used to aid in system calibration and diagnosis at a later stage when the system is up and running.

In this paper, we study the performance in terms of system throughput and delay. Also, equations are given for CPU utilization and saturation conditions. Since our analysis is based on queueing theory, our analytical work can be easily extended to study mean waiting time, mean number of packets in system and queues, blocking

probability, etc. [18]. In addition, our analytical work can be important for engineering and designing various NIC (Network Interface Card) 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 analysis for the above three models. Section 3 gives verification and validation of analysis. Finally, Section 4 concludes the study and identifies future work.

2. Analysis

In this section we present an analytical study to examine the impact of interrupt overhead on system performance. First we define the system parameters. Let λ be the mean incoming packet arrival rate, and μ be the mean protocol processing rate carried out by the kernel. Therefore $1/\mu$ is the time the system takes to process the incoming packet and deliver it to the application process. This time includes primarily the network protocol stack processing carried out by the kernel, excluding any interrupt handling. Let T_{ISR} be the interrupt handling time, which is basically the interrupt service routine time for handling incoming packets. We will also define $\rho = \lambda/\mu$. ρ is as a measure of the traffic intensity or system load. We study the system performance in terms of system throughput (γ) and delay (R). System throughput is the rate at which packets are delivered by the kernel to the application process. System delay is the latency of delivering one packet by the kernel to the application once received by the network adapter. In addition, we investigated two important and critical operating points for the system, mainly receive lock and saturation. Receive-livelock point is the point at which system throughput becomes zero. Saturation point is the point at which the system can not keep up with the offered load.

Throughout our analysis, we assume the following:

i) It is reasonable not to assume the times for protocol processing or ISR handling to be constant. Both of these service times change due to various system activities. For example ISR handling for incoming packets can be interrupted by other interrupts of higher priority, e.g. timer interrupts. Also, protocol processing can be interrupted by higher priority kernel tasks, e.g. scheduler. For our analysis, we assume both of these service times to be exponential. In Section 3, we demonstrate that this assumption gives a good 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.

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 3, it turns out that our model with the above assumptions gives a good approximation to real experimental measurements. Traffic with variable-size packets, e.g. Jumbo frames, and other traffic distributions, e.g. bursty traffic [19], are currently being studied by the authors using simulations and results are expected to be reported in the near future.

2.1. Ideal System

In the ideal system, we assume the overhead involved in generating interrupts is totally ignored. With our assumptions, we can simply model such a system as an M/M/1/B queue with a Poisson packet arrival rate λ and a mean protocol processing time of $1/\mu$ that has an exponential distribution. B is the maximum size the system buffer can hold. M/M/1/B queueing model is chosen as opposed to M/M/1 since we can have the arrival rate go beyond the service rate, i.e., $\rho > 1$. This assumption is a must for Gigabit environment where under heavy load λ can be very high compared to μ.

2.2. PIO Option

This option is for hosts which have no DMA engines or for hosts that disable the DMA. Disabling DMA can be desirable if the host's PCI bus is highly contended by other I/O devices or the incoming packets, to be transferred to host memory, arrive at a low rate. According to [20], PIO option can also be attractive when considering factors such as cost, simplicity, and speed and efficiency in copying relatively small-size packets. Also, traditional network adapters or Network Interface Cards (NICs) as those of 10Mbps Ethernet and 16Mbps Token Ring are not equipped with DMA engines. In PIO, 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. When the network packet processing is triggered, the kernel processes the incoming packet by executing the network protocol stack and delivers the packet data to user application [3].

There are two possible system delivery options of packet to 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 [6-8,14-16]. 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.

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.

Modeling Interrupts is a Challenging and Difficult Task. One may think that such an interrupt-driven system can be simply modeled as a priority queueing system 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.

We model the interrupt-driven system as an M/G/1/B queue with a Poisson packet arrival rate of λ and a mean effective service time of $1/\mu'$ that has a general distribution. The mean effective service time is the effective CPU time for packet processing carried out by the kernel's network protocol stack. We next determine the mean effective service time for such a behavior.

Let us assume copying of the packet from NIC to kernel memory is exponentially distributed with a mean service time of $1/c$. Also for simplicity, let us assume that the T_{ISR} for servicing one packet, including the copying of the packet from NIC to kernel memory, is exponentially distributed with a mean of $1/r$. One can express the mean effective service rate as

μ' = 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.

$$\mu' = \mu \cdot (\%\ \text{CPU availability for protocol processing}). \quad (1)$$

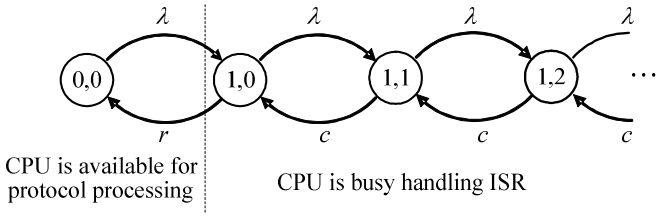

Figure 1. Markov state transition diagram for modeling CPU usage with PIO

In order to determine the CPU availability percentage for protocol processing and interrupt handling, we use a Markov process to model the CPU usage, as illustrated in Figure 1. 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 $0 \le n < \infty$ represents the state where the CPU is busy handling interrupts. *n* denotes the number of interrupts that are batched or masked off during T_{ISR}. 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 1, can be expressed as

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

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,

$$p_{0,0} = \left[1 + \frac{\lambda}{r}\sum_{n=0}^{\infty}\left(\frac{\lambda}{c}\right)^{n}\right]^{-1} = \frac{r\left(1-\frac{\lambda}{c}\right)}{r\left(1-\frac{\lambda}{c}\right)+\lambda} = \frac{rc - r\lambda}{rc - r\lambda + c\lambda}. \tag{3}$$

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

$$p_{0,0} = \begin{cases} \dfrac{rc - r\lambda}{rc - r\lambda + c\lambda}, & \lambda < c \\ 0. & \lambda \ge c \end{cases} \tag{4}$$

And hence, the mean effective service rate is expressed as

$$\mu' = \begin{cases} \mu \cdot \left(\dfrac{rc - r\lambda}{rc - r\lambda + c\lambda}\right), & \lambda < c \\ 0. & \lambda \ge c \end{cases} \tag{5}$$

It is to be noted from equation (1) that the distribution of the effective service time is exponential with a mean $1/\mu'$, as μ is multiplied by a constant fraction, that is (% CPU availability for protocol processing). Therefore, the system can be modeled as M/M/1/B queue with a mean service rate of μ'.

2.3. DMA Option

PIO-based design or configuration can be an attractive option due to cost, simplicity, and speed and efficiency in copying relatively small-size packets, as discussed in Section 2.2. However, a major drawback for PIO 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 Intel, 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. 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 for PCI 32/33 MHz bus and over 4 Gbps for PCI 64/66 MHz bus [21, 22].

The DMA-based design option can be modeled as an M/G/1/B queue with a Poisson packet arrival rate of λ and a mean effective service time of $1/\mu'$ that has a general distribution. In order to determine the mean effective service time $1/\mu'$, we need to determine the CPU availability percentage for protocol processing and interrupt handling. We use a Markov process to model the CPU usage, as illustrated in Figure 2. 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 $0 \leq n < \infty$ represents the state where the CPU is busy handling interrupts. *n* denotes the number of interrupts that are batched or masked off during T_{ISR}. 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.

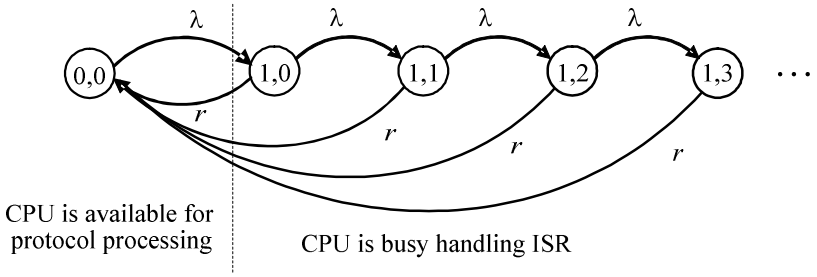

Figure 2. Markov state transition diagram for modeling CPU usage with DMA

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

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

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.

Thus, the mean effective service rate can be expressed as

$$\mu' = \mu \cdot \frac{r}{\lambda + r}. \tag{6}$$

As we already know the mean effective service time $1/\mu'$ is exponential. Please refer to equation (2). Therefore, this system can also be modeled as M/M/1/B queue with a mean service rate of μ'.

2.4. Throughput Analysis

In an M/M/1/B model or the ideal system, the mean system throughput γ can be expressed as

$$\gamma = \mu\,(1 - p_0), \tag{7}$$

where p_0 is the probability of zero packets being processed by the kernel's network protocol stack. p_0 is given by

$$p_0 = \begin{cases} \dfrac{1-\rho}{1-\rho^{B+1}} & (\rho \neq 1), \\ \dfrac{1}{B+1} & (\rho = 1). \end{cases} \tag{8}$$

For PIO, the system throughput can be expressed by equation (7). However, the mean service rate μ of equation (7) must be replaced by the mean effective service rate μ' of equation (5). Similarly, for DMA, the system throughput can be expressed by equation (7). However, the mean service rate μ of equation (7) must be replaced by the mean effective service rate μ' of equation (6).

2.5. Receive-Livelock Point

A critical operating point for a system is the condition at which the system throughput becomes zero. This condition is where receive livelock occurs. By examining equation (7), system throughput (γ) can be zero if $p_0 = 1$ or the mean service rate μ is zero.

For PIO, the receive-livelock point becomes zero when the mean effective service rate μ' of equation (5) becomes zero. Based on equation (5), the receive-livelock point depends only the values of λ and c. It precisely occurs when $\lambda \geq c$. Further, 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 mean effective service rate μ' of equation (6) depends only on the values of λ and *r*, as copying has been eliminated. From theoretical aspect as given by equation (6), μ' does not become zero unless $\lambda \to \infty$. This means that the receive-livelock point for DMA occurs *theoretically* when $\lambda \to \infty$. However from practical aspect, having an incoming packet arrival rate λ to reach infinity is not possible due to the physical limitation of host hardware as well as the capacity limit of network link. Host physical limitation is related to PCI bus, DMA engines, memory speed, etc. Nowadays the transfer bit rate of a 64/66 MHz PCI bus goes little over 4 Gbps [21,22]. Also the current exiting Ethernet links have a bandwidth of up to 10 Gbps. Of course the incoming packet rate λ is much less than such a bit rate as it gets reduced by the packet size and payload overhead. Therefore from practical aspect, the receive livelock should not occur when employing DMA.

2.6. Saturation Point

Another critical operating point for a system is the situation at which the system can not keep up with the offered network load. This is referred to as the saturation point where $\rho = 1$, or is defined as the "cliff" point of system throughput. It is where the throughput starts falling as the network load increases.

For PIO, the saturation point can be expressed as

$$\rho = 1 \quad or \quad \lambda = \mu'.$$

Solving for λ, we get

$$\lambda = \mu \cdot \left(\frac{rc - r\lambda}{rc - r\lambda + c\lambda} \right) = \mu \cdot \left(\frac{rc - r\lambda}{rc + (c - r)\lambda} \right).$$

Since the term $rc+(c-r)\lambda$ is always positive, then

$$\lambda\left(rc+(c-r)\lambda\right)=\mu(rc-r\lambda),$$
$$(c-r)\lambda^2+rc\lambda=rc\mu-r\mu\lambda,$$
$$(c-r)\lambda^2+r(c+\mu)\lambda-rc\mu=0.$$

The left hand side is a quadratic equation that has the following solutions

$$\lambda=\frac{-r(c+\mu)\pm\sqrt{r^2(c-\mu)^2+4rc^2\mu}}{2(c-r)}.$$

Since the denominator is positive then the negative sign is rejected. Thus, the saturation point can be expressed as

$$\lambda=\frac{\sqrt{r^2(c-\mu)^2+4rc^2\mu}-r(c+\mu)}{2(c-r)}. \tag{9}$$

For DMA, the saturation point can be expressed as

$$\rho=1 \quad or \quad \lambda=\mu\cdot\frac{r}{\lambda+r}.$$

Solving for λ, we get

$$\lambda(\lambda+r)=\mu r \quad \Rightarrow \quad \lambda^2+r\lambda-\mu r=0.$$

The roots of the quadratic equation $\lambda^2+r\lambda-\mu r=0$ are

$$\lambda=\frac{-r\mp\sqrt{r^2+4\mu r}}{2}=\frac{-r\mp r\sqrt{1+4\frac{\mu}{r}}}{2}.$$

Since the term under the square root is always greater than one then the negative sign is neglected. Therefore, the saturation point for this system occurs when

$$\lambda=\frac{r}{2}\left(\sqrt{1+4\frac{\mu}{r}}-1\right). \tag{10}$$

It is to be noted that equation (9) and equation (10) can also be derived by finding the maximum point of system throughput. This can be done by taking the derivative of the system throughput of equation (7) with respect to λ and setting it to zero, i.e. $\frac{d\gamma}{d\lambda}=0$, and then solving for λ.

2.7. Delay Analysis

In M/M/1/B model or the ideal system, the mean system packet processing delay $E(r)$ can be given by

$$E(r) = \frac{E(n)}{\lambda(1 - p_B)},$$

where $E(n) = \frac{\rho}{1-\rho} - \frac{B+1}{1-\rho^{B+1}}\rho^{B+1}$, $\rho = \lambda/\mu$, and

For systems with PIO and DMA, the mean system delay per packet is affected by both ISR handling and protocol processing. An incoming packet experiences a delay due to interrupt handling and due to the delay of protocol processing. To determine this delay analytically, we apply Jackson's queueing theorem in which the mean system delay is approximated to be the sum of the mean delay of interrupt handling plus the mean delay of protocol processing. Hence the total mean system delay, $E(r)$, according to Jackson theorem can be expressed as

$$E(r) = E_{ISR}(r) + E_{IP}(r), \tag{11}$$

where $E_{ISR}(r)$ is the mean delay due to ISR and $E_{IP}(r)$ is mean delay due to protocol processing.

For PIO, $E_{ISR}(r)$ can be computed using the Markov chain depicted in Figure 1. First we compute the average number of packets being serviced by one ISR, $E_{ISR}(n)$, as follows

$$E_{ISR}(n) = \sum_{n=0}^{\infty}(n+1)p_{1,n} = \sum_{n=1}^{\infty} np_{1,n} + \sum_{n=0}^{\infty} p_{1,n}.$$

When substituting $p_{1,n}$, given by equation (2), we get

$$E_{ISR}(n) = p_{0,0}\left(\frac{\lambda}{r}\right)\left[1 - \left(\frac{\lambda}{c}\right)^2\right]^{-1}.$$

$p_{0,0}$ is given by equation (3). Let $v = 1 - \frac{\lambda}{c}$. When substituting and simplifying, we get

$$E_{ISR}(n) = \frac{\lambda}{v(rv + \lambda)}.$$

Therefore the mean delay due to ISR, $E_{ISR}(r)$, can be computed using Little's law as

$$E_{ISR}(r) = \frac{1}{v(rv + \lambda)}, \tag{12}$$

Where $v = 1 - \frac{\lambda}{c}$.

As for the mean delay caused by protocol processing, $E_{IP}(r)$, in PIO, it is simply the mean delay encountered in the *M/M/1/B* queueing system with $\rho = \left(\frac{\lambda}{\mu'}\right)$. According to [18], such delay can be expressed as

$$E_{IP}(r) = \frac{E_{IP}(n)}{\lambda'}, \tag{13}$$

where $E_{IP}(n) = \frac{\rho}{1-\rho} - \frac{(B+1)\rho^{B+1}}{1-\rho^{B+1}}$ and λ' is the mean effective arrival rate. λ' is the same as the mean system throughput γ given by equation (7) with $\mu = \mu'$. In PIO, remember that the μ', to be substituted in the equations of λ' and ρ, is given by equation (5).

For DMA, $E_{ISR}(r)$ is simply $1/r$. This is so due to the nature of servicing packets during ISR handling. The mean ISR handling time for one packet or many packets is the same, i.e. $1/r$. This delay can also be computed using the Markov chain depicted in Figure 2. First we compute $p_{1,n}$ from Figure 2. Using mathematical induction and the iterative method of solving the steady-state difference equations, $p_{1,n} = \frac{r}{\lambda}\left(\frac{\lambda}{\lambda + r}\right)^{n+2}$. The average number of packets being serviced by one ISR, $E_{ISR}(n)$, can be expressed in a similar fashion as in PIO where by

$$E_{ISR}(n) = \sum_{n=0}^{\infty}(n+1)p_{1,n} = \sum_{n=1}^{\infty} np_{1,n} + \sum_{n=0}^{\infty} p_{1,n}\ .$$

With further simplification,

$$E_{ISR}(n) = \frac{\lambda}{r}.$$

And therefore, the average ISR delay per packet, $E_{ISR}(r)$, according to Little's law, is

$$E_{ISR}(r) = \frac{E_{ISR}(n)}{\lambda} = \frac{1}{r}. \tag{14}$$

As for the mean delay caused by protocol processing, $E_{IP}(r)$, in DMA, it is simply the mean delay encountered in the *M/M/1/B* queueing system with $\rho = \left(\frac{\lambda}{\mu'}\right)$. $E_{IP}(r)$ is expressed in equation (13). It is to be

noted that in DMA, λ' is the same as the mean system throughput γ given in equation (7) with $\mu = \mu'$. Also remember that the μ', to be substituted in the equations of λ' and ρ, is given by equation (6) in this case.

3. Analysis Verification and Validation

In this section we verify our analysis two ways: by using simulation and by considering some special cases. As for validation, we compare our analysis results to reported experimental measurements in the literature.

3.1. Special Cases

We consider a special case when interrupt handling is ignored, i.e., the ideal system when $T_{ISR} = 0$. In this situation when $T_{ISR} = 0$, $r \to \infty$ and $c \to \infty$. With this condition, we prove that the equations for PIO and DMA mean effective service rate, saturation point and mean system latency become the same equations for the ideal system.

Mean effective service time. We prove that equations (5) and (6) yield the ideal system mean service rate μ as follows:

For PIO mean effective service rate of equation (5),

$$\mu' = \lim_{r,c\to\infty} \mu \cdot \left(\frac{rc - r\lambda}{rc - r\lambda + c\lambda} \right)$$
$$= \lim_{r,c\to\infty} \mu \cdot \left(\frac{1 - \lambda/c}{1 - \lambda/c + \lambda/r} \right) = \mu .$$

For DMA mean effective service rate of equation (6),

$$\mu' = \lim_{r\to\infty} \mu \cdot \left(\frac{r}{\lambda + r} \right) = \lim_{r\to\infty} \mu \cdot \left(\frac{1}{\lambda/r + 1} \right) = \mu .$$

Saturation Point. We prove that equations (9) and (10) yield the ideal system saturation point of $\lambda = \mu$ as follows:

For PIO saturation point of equation (9), we first take the limit as $r \to \infty$, and then multiply both numerator and denominator by $1/r$.

$$\lambda = \lim_{r\to\infty} \frac{\sqrt{r^2(c-\mu)^2 + 4rc^2\mu} - r(c+\mu)}{2(c-r)}$$

$$\lambda = \lim_{r\to\infty} \frac{\sqrt{(c-\mu)^2 + 4\frac{c^2}{r}\mu} - (c+\mu)}{2(\frac{c}{r}-1)}$$

$$\lambda = -\frac{1}{2}\sqrt{(c-\mu)^2} + \frac{1}{2}(c+\mu)$$

$$\lambda = -\frac{1}{2}[(c-\mu)-(c+\mu)] = \mu.$$

Now we take the limit as $c \to \infty$. But this is still μ as it does not depend on *c*.

For DMA saturation point of equation (10),

$$\lambda = \lim_{r\to\infty}\left(\frac{r}{2}\sqrt{1+4\frac{\mu}{r}} - \frac{r}{2}\right) = \lim_{r\to\infty}\left(\frac{\sqrt{1+4\frac{\mu}{r}} - 1}{\frac{2}{r}}\right).$$

Applying L'Hopital Rule, we get

$$\lambda = \lim_{r\to\infty}\left(\frac{-2\mu}{r^2\sqrt{1+4\mu/r}} \Big/ \frac{-2}{r^2}\right) = \lim_{r\to\infty}\left(\frac{\mu}{\sqrt{1+2\mu/r}}\right) = \mu.$$

Mean system delay. When the interrupt is ignored, it is expected the delays encountered by the interrupt overhead resort to zero. When examining $E_{ISR}(r)$ for PIO and DMA in equation (12) and equation (14), respectively, it can be concluded easily that $E_{ISR}(r)$, when $r \to \infty$ and $c \to \infty$, becomes zero. This is in line with intuition and our expectation. Consequently, the mean system delay becomes that of an M/M/1/B queueing system, i.e., the ideal system.

3.2. Simulation

In order to verify and validate our analytical models, a discrete-event simulation model was developed and written in C language. A detailed description and flowcharts of the simulation model for normal interruption can be found in [23]. The assumptions of analysis were used. The simulation followed closely and carefully the guidelines given in [24]. We used the PMMLCG as our random number generator [24]. The simulation was automated to produce independent replications with different initial seeds that were ten million apart. During the simulation run, we checked for overlapping in the random number streams and ascertain that such a condition did not exist. The simulation was terminated when achieving a precision of no more than 10% of the mean with a confidence of 95%. We employed and implemented dynamically the *replication/deletion* approach for means discussed in [24]. We computed the length of the initial transient period using the MCR

(Marginal Confidence Rule) heuristic developed by White [25]. Each replication run lasts for five times of the length of the initial transient period. Analytical and simulation results, as will be demonstrated in Section 3, were very much in line.

3.3. Numerical Examples

In this section, we report and compare results of system throughput and delay for the ideal system, PIO and DMA. For validation, we compare our analysis and simulation results of system throughput 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 engines. 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 ($1/\mu$), which included copying of packet data to user space, was 150 μ seconds.

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 ($1/\mu$) was 9.7 μ seconds. In all of our examples, we fix the kernel's protocol processing buffer B to a size of 1024 packets, which occupies about 1.6M bytes of host memory when assuming a maximum of 1536 bytes per packet in accordance to IEEE802.3 standards. This size is typically a power of 2 and a configurable parameter [26].

Figure 3 and 4 show four overlaid plots of system throughput as a function of packet arrival rate. For verification purposes, the assumptions of analysis were used for simulation. The four plots are as follows:

i. Ideal system. The results are from both analysis and simulation when ignoring interrupt overhead, i.e., the mean service time for ISR ($1/r$) is 0.
ii. Experimental. The results are those of the PIO and DMA experimental measurements as reported in [3] and [4], respectively.
iii. Analysis. The results are obtained by our analysis given the same experimental parameters of [3] and [4]. Poisson arrival is assumed here, i.e., the inter-arrival times of packets are exponentially distributed.
iv. Simulation with Poisson arrival. The results are obtained by the discrete-event simulation given the experimental parameters of [3] and [4]. As in analysis, the incoming packets follow a Poisson process.

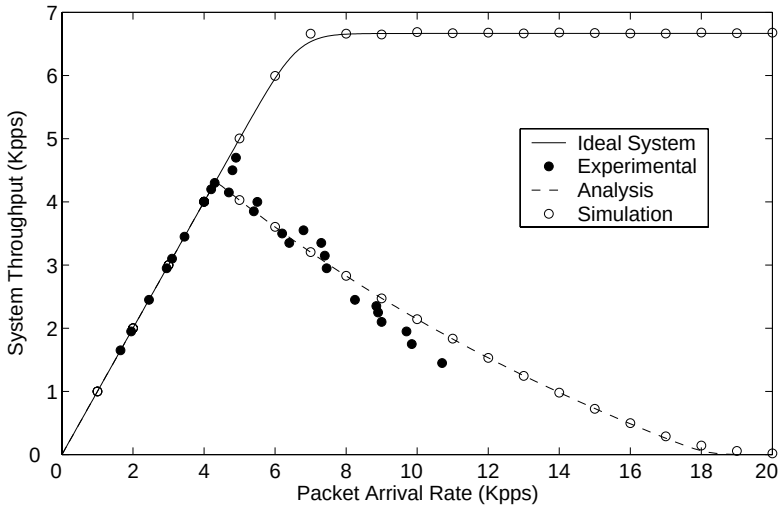

Figure 3. System throughput with PIO

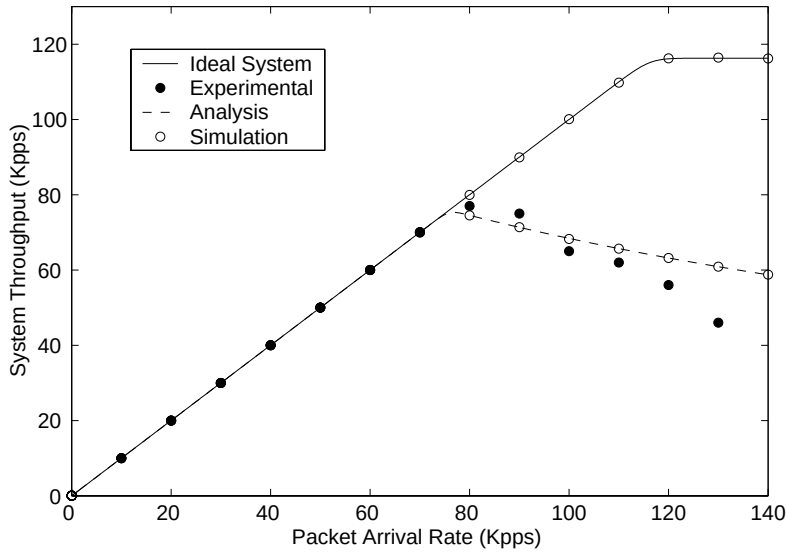

Figure 4. System throughput with DMA

By subjectively eyeballing the graphs in Figure 3 and Figure 4, it is clear that the results of analysis are in a perfect accordance to those of simulation. Therefore, our analysis is correct and verified. As for validation, we compare the experimental results to those of analysis. We see that the analysis results give adequate approximation to real experimental measurements. Both figures depict that the analysis results match exactly those of experimental at light load, just before the system saturation point or throughput cliff point. 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. When comparing PIO and DMA system throughput, 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 receive-livelock point for PIO occurs at $\lambda = c = 18{,}182$ pps. On the other hand for DMA, the receive-livelock point is not shown within the scope of the network offered load. As for DMA and as discussed in Section 2.5, the receive-livelock point does not occur.

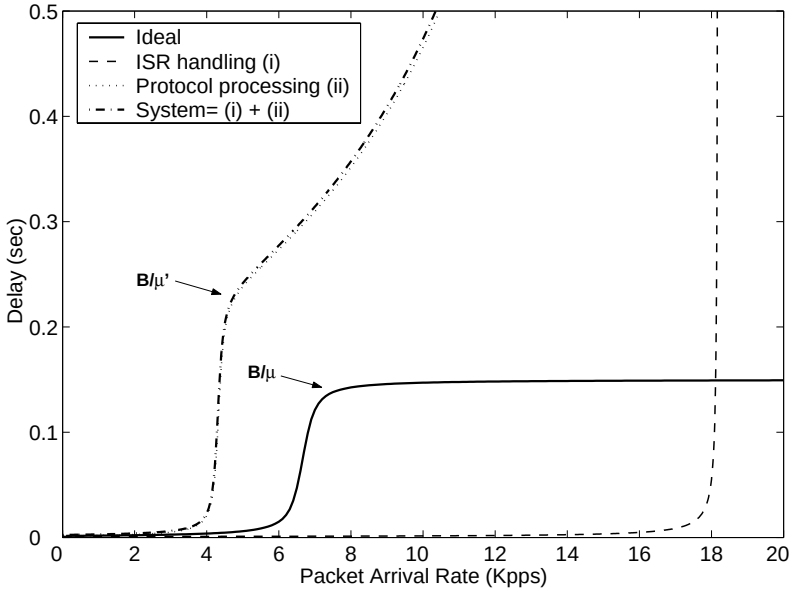

Figure 5. Mean delays with PIO

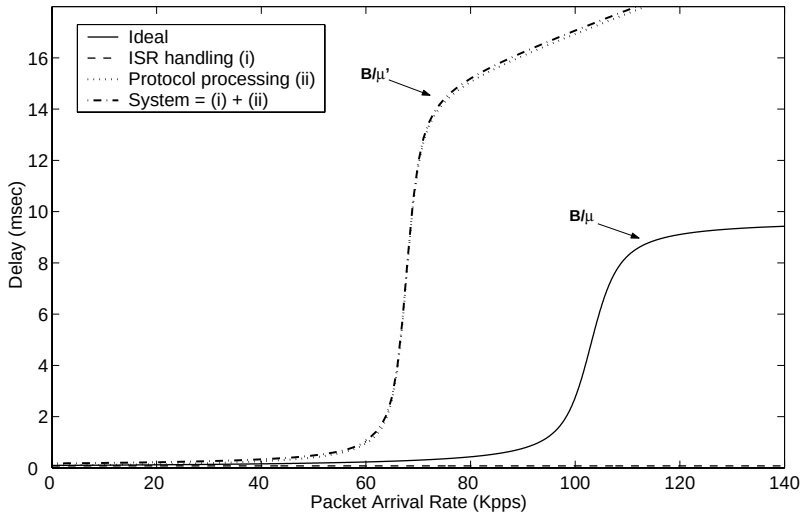

Figure 6. Mean delays with DMA

Figure 5 and Figure 6 show the mean delays for the same offered load for PIO and DMA, respectively. Also the corresponding mean system delays for the ideal system delay are shown. The discrete-event simulation results were very much in line with those of analysis, and not shown in the figures for clarity. Delays using real experiments were not measured in [3] and [4]. The figures show the mean delays for ISR handling, protocol processing, and system. As illustrated by analysis the system delay is the sum of ISR delay and protocol processing delay. In general, both figures show that at light load the mean system latency is small but increases considerably when the system is overloaded. This will result in an undesirable excessive latency in which packets start timing out. It is observed from both figures that in general the system delay is dominated by the protocol processing delay than the ISR handling. ISR handling delay, given in this numerical example, is small compared to protocol processing delay. However, as shown in Figure 5 for PIO, ISR handling delay shoots to infinity at only a very high load. For DMA, shown in Figure 6, the ISR handling delay remains constant all the time.

The mean protocol processing delay and system delay take a different shape than that of ISR handling. For the ideal system of the PIO example, Figure 5 shows the mean system delay reaches the cliff point of an arrival rate of 6,667 pps, and then the delay sharply increases until it reaches a high value. After that the mean system delay stays flat. The high value is determined by the size of the queue and the mean service rate, and it is equal to B/μ. For ideal system this value is 150 msec. Figure 5 shows that the mean delay also increases sharply when reaching the cliff point of an arrival rate of 4,328 pps. The delay then does not flatten off at B/μ', but rather it slowly shoots to infinity. This is because as the arrival rate increases right after the cliff point, the mean effective service rate μ' decreases rapidly to zero, and thus resulting in a very huge delay.

For DMA, Figure 6 shows the mean ideal system delay sharply increases when reaching the cliff point of an arrival rate of 103,093 pps and then it stays flat after that with a mean delay value of 9.7 msec. Figure 6 also depicts that the mean system delay sharply increases when reaching the cliff point of an arrival rate of 67,750 pps and then it starts increasing considerably as the mean effective service rate μ' decreases.

4. Conclusion

We developed analytical models to study the impact of interrupt overhead caused by high-speed network traffic on OS performance. We presented a throughput-delay analysis for hosts when subjected to light and heavy network loads. We considered high-speed network hosts with PIO and DMA design options or configurations. We also investigated two critical system operating points, mainly receive livelock and saturation. As noted, degraded throughput and excessive latency can be encountered due to heavy network loads. If the system is poorly designed or configured, 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 throughput and delay can be reached by choosing between DMA vs. PIO configurations 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 system throughput and latency. 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, degradation in system throughput is far worse in PIO than DMA. As opposed to PIO, the receive livelock point does not occur in DMA. Therefore, utilizing DMA engines for high-speed 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.

Acknowledgement

The authors acknowledge the support of King Fahd University of Petroleum and Minerals in the development of this work.

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] C. Traw, and J. Smith, "Giving Applications Access to Gb/s Networking," IEEE Network, vol. 7, no. 4, July 1993, pp. 44-52.

[14] 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.

[15] 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.

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

[17] 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.

[18] L. Kleinrock, Queueing Systems: Theory, vol 1, Wiley, 1975.

[19] 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.

[20] 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.

[21] K. Kochetkov, “Intel PRO/1000 T Desktop Adapter Review,” http://www.digit-life.com/articles/intelpro1000t

[22] 3Com Corporation, "Gigabit Server Network Interface Cards 7100xx Family,” http://www.costcentral.com/pdf/DS/3COMBC/DS3COMBC109285.PDF

[23] K. Salah and K. El-Badawi, “Analysis and Simulation of Interrupt Overhead Impact on OS Throughput in High-Speed Networks,” *International Journal of Communication Systems*, vol. 18, no. 5, Wiley Publisher, June 2005, pp. 501-526

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

[25] J. White, “An Effective Truncation Heuristic for Bias Reduction in Simulation Output,” *Simulation Journal*, vol. 69, no. 6, December 1997, pp. 323-334

[26] A. Rubini and J. Corbet, “The Linux Device Drivers,” O’Reilly, 2001.