Analysis and Simulation of Interrupt Overhead Impact on OS Throughput in High-Speed Networks

K. Salah 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

The paper presents analytical and simulation models to study the impact of interrupt overhead on operating system throughput of network elements such as PC-based routers, servers, and hosts when subjected to high-speed network traffic. Under such high network traffic, the system throughput will be negatively affected due to interrupt overhead caused by the incoming traffic. 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 throughput. One model is for employing PIO in which network adapters are not equipped with DMA engines, and the other model is for employing DMA in which network adapters are equipped with DMA engines. The paper also describes detailed discrete-event simulation models for the ideal system and for systems with DMA and PIO. Simulations results as well as reported experimental measurements show that our analytical models are valid and give a good approximation. Our analysis and simulation work can be valuable in providing insight to understand and predict system behavior, as well as improving and maintaining good host performance. The paper identifies analytically critical design operation points such as that of overload condition. The paper also proposes solutions and recommendations for improving performance.

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

1. Introduction

Interrupt overhead of high-speed network devices can have a significant negative impact on system performance. 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.

These days we have a widespread deployment and development of high-speed network devices. In particular, a massive deployment has taken place for 1-Gigabit and 10-Gigabit for Ethernet network devices. However, existing operating systems still use for the most part the same mechanisms to handle both network processing and traditional I/O devices. The shift to higher packet arrival rate can subject a host to congestive collapse.

Interrupt-driven systems tend to perform very badly under such heavy load conditions. 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 and latency 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 [1,3,5-13] 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 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 the 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, interrupt handling 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 fail to identify the saturation point of the host.

In [17], 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 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 throughput. One model is for employing PIO in which network adapters are not equipped with DMA engines, and the other model is for employing DMA 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.

In this paper, we study the performance in terms of system throughput. Also, equations are given for CPU utilization, CPU availability, and stability conditions. Since our analysis is based on queueing theory, our analytical work can be easily extended to study mean system delay, 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 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 describes the receive livelock phenomenon reported in literature. Section 3 presents modeling and analysis for the ideal system and systems with PIO and DMA. Section 4 gives verification and validation of analysis and describes detailed discrete event simulations models for ideal, PIO, and DMA systems. Section 5 discusses results and proposes recommendations for improving system performance. Finally, Section 6 concludes the study and identifies future work.

2. Receive Livelock

In this section we briefly describe the phenomenon of receive livelock. Incoming network packets received at a host must either be forwarded to other hosts, as is the case in PC-based routers, or to application processes where they are consumed. The delivered system throughput is a measure of the rate at which such packets are processed successfully. Figure 1, adopted from [2,3], shows the delivered system throughput as a function of offered input load. Please note that the figure illustrates conceptually the expected behavior of the system and does not illustrate analytical behavior. The figure illustrates that in the ideal case, no matter what the packet arrival rate, every incoming packet is processed. However, all practical systems have finite processing capacity, and cannot receive and process packets beyond a maximum rate. This rate is called the Maximum Loss-Free Receive Rate (MLFRR) [2]. Such rate is an acceptable rate and is relatively flat after that.

Under network input overload, a host can be swamped with incoming packets to the extent that the effective system throughput falls to zero. Such a situation, where a host has not crashed but is unable to perform useful work, such as delivering received packets to user processes or running other ready processes, is known as *receive livelock.* Similarly, under receive livelock, a PC-based router would be unable to forward packets to the outgoing interfaces.

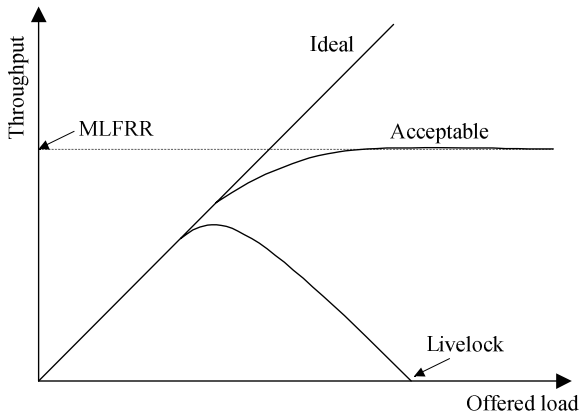

Figure 1. Receive livelock phenomenon

The main reason for receive livelock is that interrupts are handled at a very high priority level, higher than software interrupts or input threads that process the packet further up the protocol stack. At low packet arrival rates, this design allows the kernel to process the interrupt of the incoming packet almost immediately, freeing up CPU processing power for other user tasks or threads before the arrival of the next packet. However, if another packet arrives before the completion of handling the first one (e.g., in the case of high packet arrival rate), starvation will occur for user tasks and threads resulting in unpleasant performance of dropping packets due to queue overflows, excessive network latency, and bad system throughput.

3. 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 packet. 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. System throughput (γ) is the rate at which packets are delivered by the kernel to the application process.

Throughout our analysis, we assume the following:

i) It is sensible not to assume the protocol processing time and ISR handling times 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 4, 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.

3.1. 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 [19], are currently being studied by the authors using simulations and results are expected to be reported in the near future.

3.2. Ideal System

This section presents analysis for the ideal situation in which the overhead involved in generating interrupts is totally ignored. Assuming packets are all of fixed size, 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.

In M/M/1/B model, the system throughput can be expressed as

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

where p_0 is the probability that the system is idle and given by

$$p_0 = \begin{cases} \dfrac{1-\rho}{1-\rho^{B+1}} & (\rho \neq 1), \\ \dfrac{1}{B+1} & (\rho = 1). \end{cases} \tag{2}$$

The available or leftover CPU time percentage for other processing, including user applications, in the ideal system, is basically $1-\rho$. This is true since the CPU utilization for ISR handling is zero, and the CPU utilization for protocol processing is $\rho = \lambda/\mu$.

3.3. Employing PIO

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. 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 a technique called zero copy [6-8,13-16]. The kernel is written such that the packet is delivered to the application using pointer manipulations. Our analytical model captures both options whereby there is only a difference 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.

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 constitutes that for ISRs and has the higher priority. The second arrival is the arrival for 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 queueing based 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.

As illustrated in Figure 2, the effective service time is the actual time available for processing a packet by the kernel's protocol stack including any T_{ISR} disruption in between. The available service time is the available time between successive T_{ISR}'s. If a packet or multiple packets arrive during such T_{ISR}, T_{ISR} will be extended to perform one copy for each of these arrived packets, as shown in Figure 2. As shown in the figure, the ISR handling for packet 3 will be extended to perform individual copying of the two packets 4 and 5 which arrived during the ISR handling of packet 3.

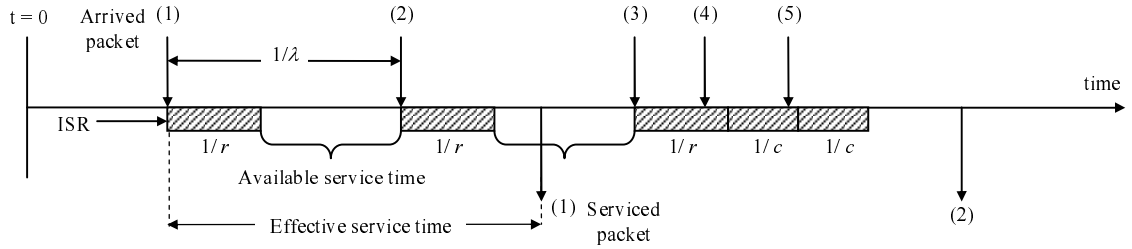

Figure 2. Effective service time with PIO

As more packets arrive during ISR handling, i.e. the arrival rate λ becomes high, most of the CPU cycles 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.

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. In other words, we have

$$\mu' = \mu \cdot (\%\ \text{CPU availability for protocol processing}). \quad (3)$$

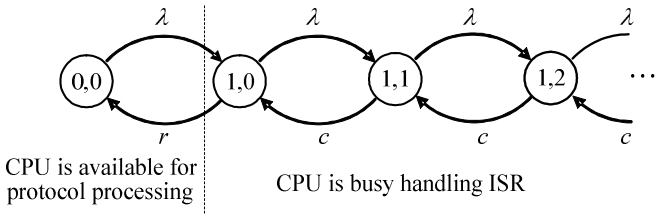

Figure 3. Markov state transition diagram for modeling CPU usage with PIO

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 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 packet arrivals or interrupts that are batched or masked off during T_{ISR}. 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 3, can be expressed as

$$p_{1,n} = \frac{\lambda^{n+1}}{r\,c^{n}}\,p_{0,0} \qquad (n = 0,1,2,\cdots). \tag{4}$$

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,

$$p_{0,0} = \left[1 + \frac{\lambda}{r}\sum_{n=0}^{\infty}\left(\frac{\lambda}{c}\right)^{n}\right]^{-1} = \frac{rc - r\lambda}{rc - r\lambda + c\lambda}. \tag{5}$$

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

$$p_{0,0} = \begin{cases} \dfrac{rc - r\lambda}{rc - r\lambda + c\lambda}, & \lambda < c \\ 0. & \lambda \geq c \end{cases} \tag{6}$$

And hence, the mean effective service rate is expressed as

$$\mu' = \begin{cases} \mu \cdot \left(\dfrac{rc - r\lambda}{rc - r\lambda + c\lambda}\right), & \lambda < c \\ 0. & \lambda \geq c \end{cases} \tag{7}$$

It is to be noted from equation (3) that the mean effective service time $1/\mu'$ is exponential. Therefore, the system can be modeled as M/M/1/B queue as is the case for the ideal system. Therefore, the system throughput can be expressed by equation (1). However, the mean service rate μ for equation (1) will have to be replaced by the mean effective service rate μ' of equation (7).

The available CPU time percentage for other processing, including user applications, when employing PIO, 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 particular point of interest is finding the stability condition for the system. The stability condition is the situation where $\rho < 1$, or is defined as the "cliff" point of system throughput. This point is also called the saturation point. It is where the throughput starts falling to zero as the system load increases. The stability condition for this model can be expressed as

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

Solving for λ, we get

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

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

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

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 or cliff point can be expressed as

$$\lambda_{cliff} = \frac{\sqrt{r^2(c - \mu)^2 + 4rc^2\mu} - r(c + \mu)}{2(c - r)}. \tag{8}$$

3.4. Employing DMA

In order to minimize CPU cycles consumed in copying packets from the NIC to kernel memory, 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.

Figure 4 shows a typical system architecture model. The figure also shows the flow path of an incoming packet between the NIC, host memory, and application. The packet is moved from the NIC Rx buffer, through the bus interface such as the PCI, to the Rx system buffer ring in the host memory, and then to the user application. As shown at initialization, the descriptor of the system Rx buffer ring is loaded into the DMA engine. The descriptor is a circular linked list of basically packet descriptors. Each packet descriptor contains the start address of the packet and its corresponding length. The length field of each packet is updated by the Rx DMA engine after the packet is copied into the host memory.

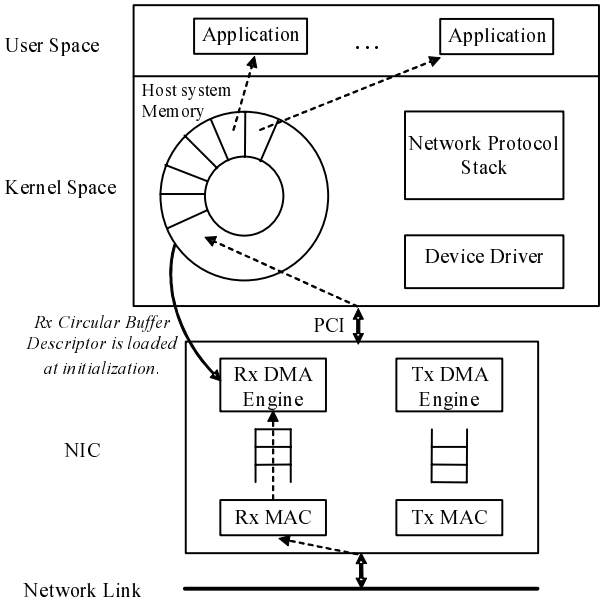

Figure 4. System Architecture Model and Flow of Arrived Packets

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 completely moved 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. Figure 5 illustrates the ISR handling for incoming packets 3 and 4 is only the ISR handling for packet 3.

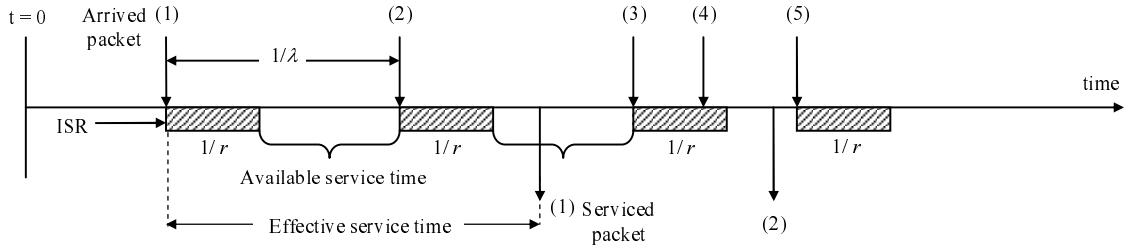

Figure 5. Effective service time with DMA

This system can also 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 6. 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 packet arrivals or interrupts that are batched or masked off during T_{ISR}. Note that 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.

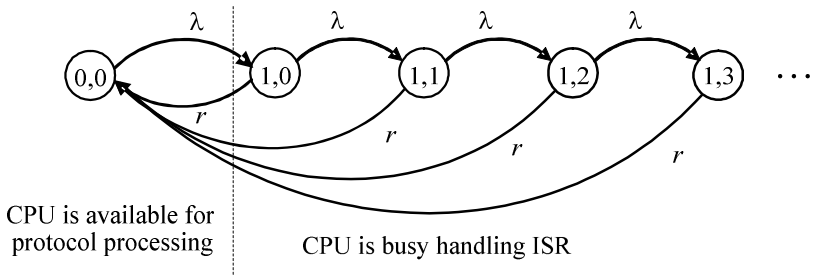

Figure 6. Markov state transition diagram for modeling CPU usage with DMA

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 handling interrupts are $r/(\lambda + r)$ and $\lambda/(\lambda + r)$, respectively.

Thus, the mean effective service rate can be expressed as

$$\mu' = \mu \cdot \frac{r}{\lambda + r}. \tag{9}$$

Similar to analysis presented in Section 3.2 for determining the system throughput when using PIO, this system can also be modeled as M/M/1/B queue as is the case for the ideal system. The system throughput can be expressed by equation (1). However, the mean service rate μ for equation (1) will have to be replaced by the mean effective service rate μ' of equation (9).

Similar to PIO analysis, the available CPU time percentage for other processing, including user applications, when employing DMA, 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 $p_{0,0} \cdot p_0$.

Also, the stability condition for this model 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 or cliff point can expressed as

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

4. Analysis Verification and Validation

In this section we verify and validate our analysis. The verification is accomplished by simulations and by considering some special cases. For validation, we compare our analysis results to two reported experimental measurements.

4.1. 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 $c \to \infty$. We prove that the equations for the PIO and DMA mean effective service rate and stability condition yield the same equations for the mean service rate and stability condition of the ideal system.

Mean effective service time. We prove that equations (7) and (9) yield the ideal system mean service rate μ as follows:

For PIO mean effective service rate of equation (7),

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

For DMA mean effective service rate of equation (9),

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

Stability Condition. We prove that equations (8) and (10) yield the ideal system stability condition of $\lambda < \mu$ as follows:

For PIO stability condition of equation (8), 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 take the limit as $c \to \infty$. But this is still μ as it does not depend on c.

For DMA stability condition 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

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

4.2. Simulation

In order to verify the analytical models, a discrete-event simulation model was developed. The implementation was carried out in C language. The simulation followed closely and carefully the guidelines given in [20]. We used the PMMLCG as our random number generator [20]. The simulation was automated to produce independent replications with different initial seeds that were one million apart. The initial seeds for the simulation model random variables within each replication were chosen to be 500,000 apart. During the simulation run, we checked for overlapping 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 90%. We employed and implemented dynamically the *replication/deletion* approach for means discussed in [20]. We computed the length of the initial transient period using the MCR (Marginal Confidence Rule) heuristic developed by White [21]. Each replication run lasts for five times of the length of the initial transient period. Figure 7 shows the general flowchart of the simulation model that is applied to the ideal system, and systems with PIO and DMA. The difference is in the type of events to be handled for each system.

Table 1. Simulation events

Event	Description
ARRIVAL	Occurs when a new packet arrives to the 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 the 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 when the CPU is not handling ISR, the NIC buffer is empty, and the 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 when the CPU is not handling ISR, the IP buffer is empty, and application buffer has packets. APP_DEPART indicates the completion of handling a packet by the user application.

Before diving into the details of simulation logic, we discuss the main components used in the simulation model:

1. *Events*: Our simulation model has three events for the ideal system: ARRIVAL, IP_DEPART, and APP_DEPART. For PIO, there are five events: ARRIVAL, ISR, ISR_COPY, IP_DEPART, and APP_DEPART. And for DMA, there are four events: ARRIVAL, ISR, IP_DEPART, and APP_DEPART. IP_DEPART and APP_DEPART are the only two events that are handled exactly for all three systems. Table 1 describes briefly these events. All these events are generated independently. This means that each event has its own seed and random-number stream.
2. *Event fields*: 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.
3. *Statistical variables*: Several statistical variables to measure system performance are used in the simulation model. Some of these of interest, and investigated in this paper, include CPU utilization, CPU availability, average number of packets in the system, and total number of packets departs the system.
4. *Queues*: The simulation has two types of queues: priority and FIFO. A priority queue, which is time based, is used by the scheduler to process next events. In addition, the simulation has three buffers implemented as FIFO queues: NIC, IP, and application. FIFO queues are also used to record the time of the packet arrival currently in the system. These times are used for statistics gathering.

Next we discuss the simulation logic used for the general simulation model. The details of the model are given in Figure 7. The simulation starts by initializing all system components. The next event is selected from the scheduler's event priority queue and the simulation clock is advanced to the time of the selected event. This latter step is performed by the scheduler. Consequently, the statistical variables are updated. Then, the type of the next event is checked and the appropriate event handler is invoked. The handling of these events depends on the system to be modeled: ideal, PIO, or DMA. Finally, the simulation ends when the total number of generated events reaches five millions.

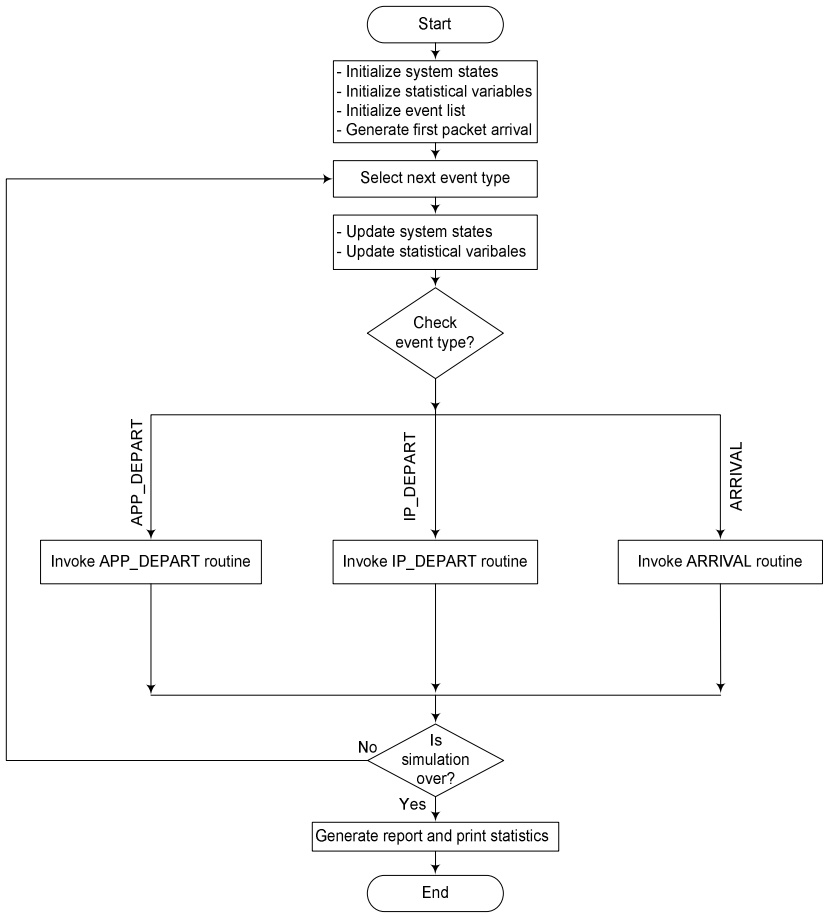

Figure 7. Flowchart of the general simulation model

Ideal System Events. The flowchart of the event handlers for the ideal system is shown in Figure 8. The simulation logic for ARRIVAL, IP_DEPART, and APP_DEPART event handling is as follows. For the ARRIVAL event, we first schedule the next ARRIVAL event. Second we increment the number of packet arrivals by one. Next we preempt or suspend any lower priority task that is currently running. In this case, the lower priority task would be only that of the user application. We then generate the IP_DEPART event and set its state to busy. Finally, we make the CPU busy processing IP_DEPART.

For IP_DEPART event handling, we first remove the packet from the IP buffer and decrement the number of packets in the IP buffer by one. Second, we append the packet to the application buffer and increment its count by one. We next check whether the IP buffer is empty or not. If the IP buffer is not empty, then we continue processing IP packets by generating the next IP_DEPART event before the routine's return. If the IP buffer is empty, we stop processing IP packets and resume any suspended user application processing. If there is no suspended user application, we notify the application to start processing the appended packet in its buffer by generating the APP_DEPART event.

For APP_DEPART event handling, we first remove the packet from the application buffer and decrement the number of packets in the application buffer by one. We next check if the application buffer is empty. If the buffer is empty, we remove the APP_DEPART event from the scheduler and make the CPU idle. If the buffer is not empty, the application stays processing packets and thus the APP_DERPART event is generated.

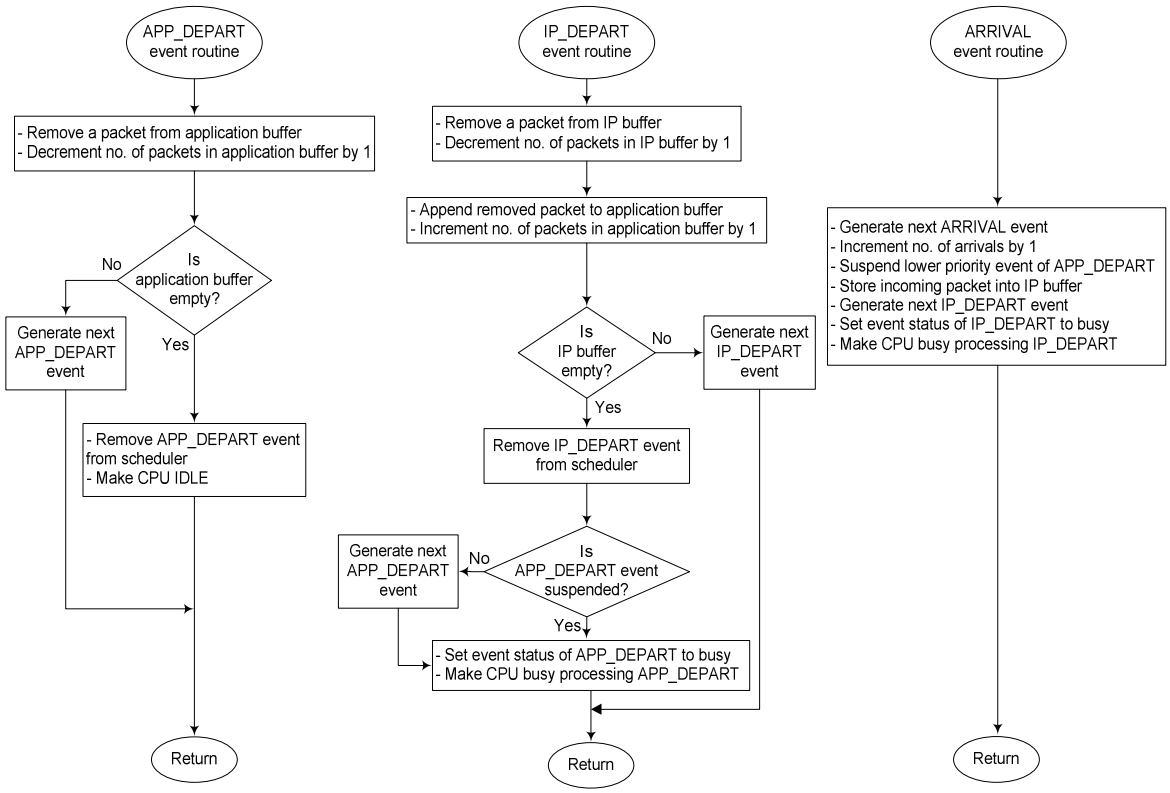

Figure 8. Flowchart of the ARRIVAL, IP_DEPART, and APP_DEPART event routines for ideal system

PIO Events. IP_DEPART and APP_DEPART events are handled exactly the same as in the ideal system. The flowchart for PIO ARRIVAL, ISR, and ISR_COPY event handling is depicted in Figure 9. For the ARRIVAL event, we first schedule the next ARRIVAL event. Second we increment the number of packet arrivals by one. Next we check if the CPU is already performing ISR handling. If so, we only store the incoming packet into NIC buffer and return. If not, the current process running on the CPU is a lower priority and has to be preempted or suspended. The lower priority process can be either IP processing or user application. We then generate the next ISR event, make CPU busy handling ISR, and finally store the incoming packet into the NIC buffer.

For the ISR event handling, we simply remove the ISR event from the scheduler and generate the next ISR_COPY event to copy the packet from NIC buffer to the IP buffer. For ISR_COPY event handling, we first copy the packet from the NIC into the IP buffer. Next, we check whether the NIC buffer is empty or not. If not empty, we continue copying by generating the next ISR_COPY event and return; otherwise, we are done

with copying. When done with copying, the ISR_COPY event is removed from the scheduler and the IP processing is resumed if it was suspended due to ISR and ISR_COPY handling. If IP processing was not suspended, IP processing is notified of the arrival of a new packet into its buffer. This notification is done by generating the next IP_DEPART event.

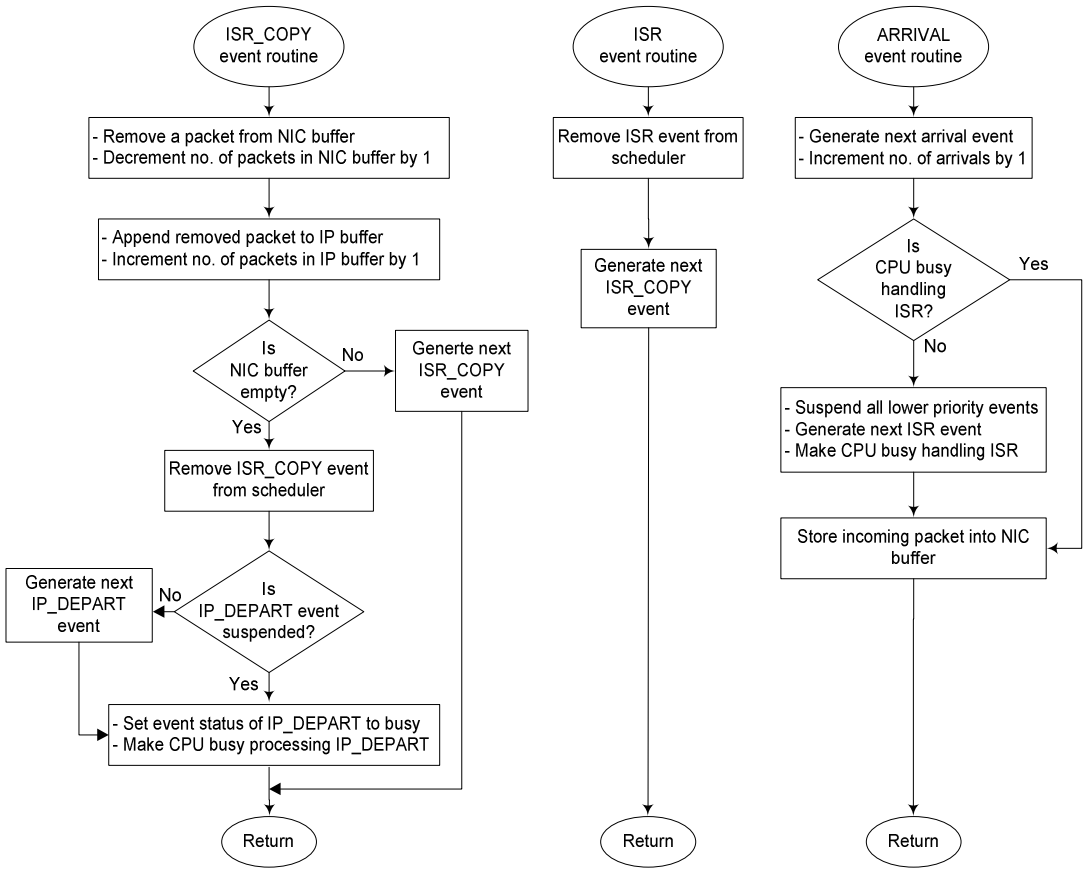

Figure 9. Flowchart of the ARRIVAL, ISR, and ISR_COPY event routines for PIO

DMA Events. Similar to PIO, IP_DEPART and APP_DEPART events are handled exactly the same as in the ideal system. The flowchart for DMA ARRIVAL and ISR event handling is depicted in Figure 10. For the ARRIVAL event, the handling is very similar to PIO except for the last step. In DMA, the storing of the incoming packet is placed directly into the IP buffer, rather than the NIC buffer. As for ISR event handling, we first remove the ISR event from the scheduler and resume IP processing if it was suspended due to ISR handling. If IP processing was not suspended, IP processing is notified of the arrival of a new packet into its buffer. This notification is done by generating the next IP_DEPART event.

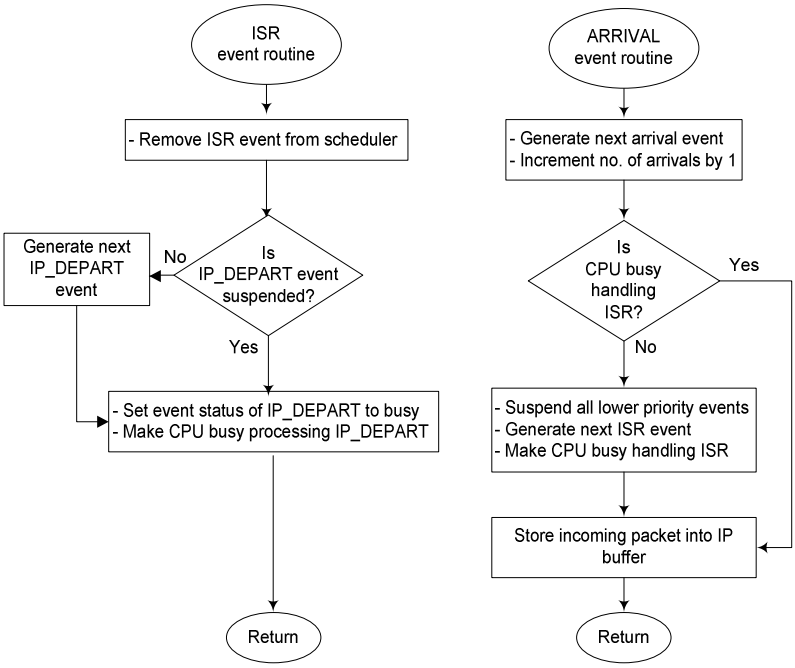

Figure 10. Flowchart of the ARRIVAL and ISR event routines for DMA

4.3. Numerical Examples

In this section, we report and compare results obtained by real experimental measurements, analysis, and simulation for the ideal system, PIO, and DMA. For validation, we compare our analysis and simulation results 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 ($1/\mu$), 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.

For all analysis and simulation runs, we fix B to a size of 1000. Figure 11 and 12 show five overlaid plots of system throughput as a function of packet arrival rate. For the simulation runs, we consider two type of packet arrivals: Poisson and constant. With Poisson, as assumed by the analysis, the inter-arrival times of packets is exponentially distributed; while with constant, the inter-arrival time of packets is fixed. We study the constant

rate because it is closer in reality to the traffic generation characteristics used by the experiments in [3] and [4] as packets are generated back-to-back with almost a constant rate. The five 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.

v. Simulation with constant arrival. The results are obtained by simulation given the experimental parameters. However, the inter-arrival times of packets are not exponential, but constant.

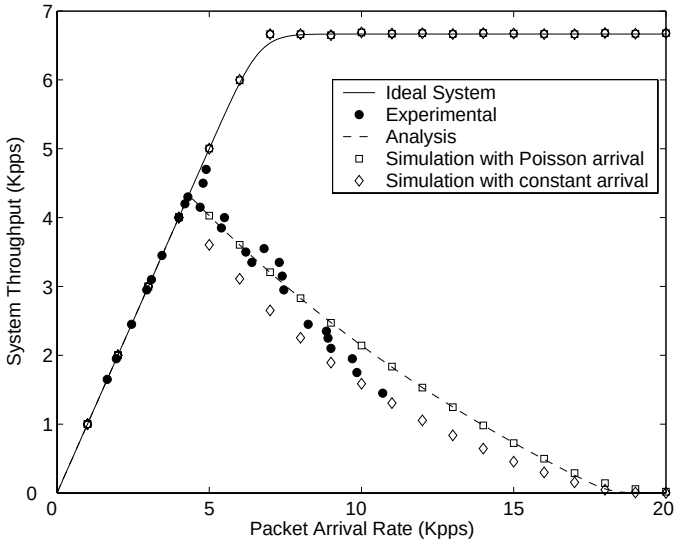

Figure 11. System throughput with PIO

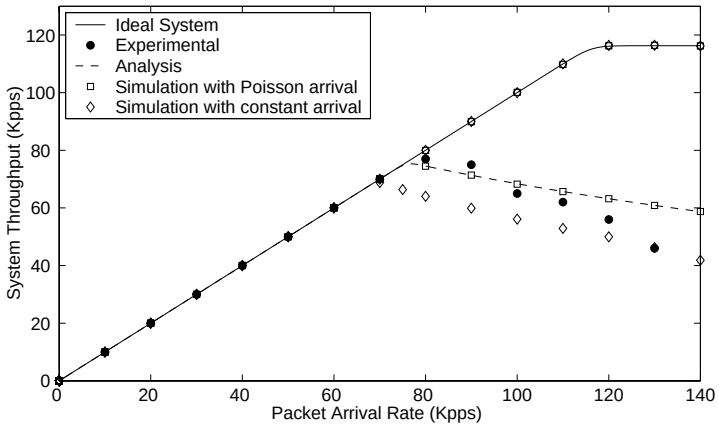

Figure 12. System throughput with DMA

Both Figure 11 and Figure 12 show that the results of analysis are in a perfect accordance to those of simulation when considering Poisson arrival for Ideal system, PIO, and DMA. 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 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. Our analysis assumed a Poisson arrival; however, as stated earlier, the inter-arrival times of the packets are not purely exponential. Remember that the packets were generated back-to-back by another host with almost a constant rate. For this purpose we also study and show the simulation results with constant inter-arrival times for incoming packets. Another factor that may contribute to the difference of experimental and analysis or simulations results at high load can be linked to internal caching and many system variables and activities. These activities can include scheduling, IP protocol processing of error checking, queuing and dropping policies, etc.

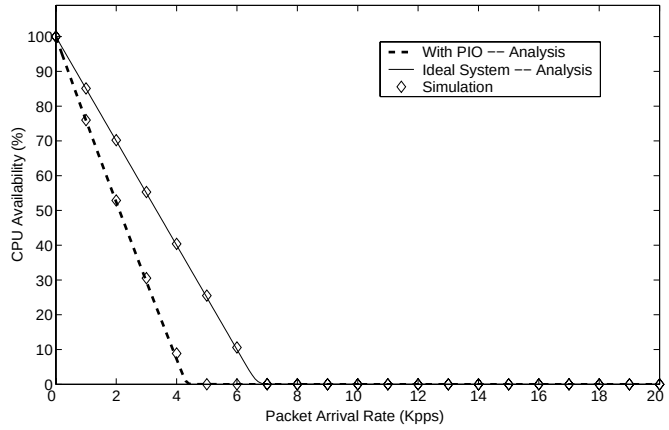

Figure 13. CPU Availability with PIO

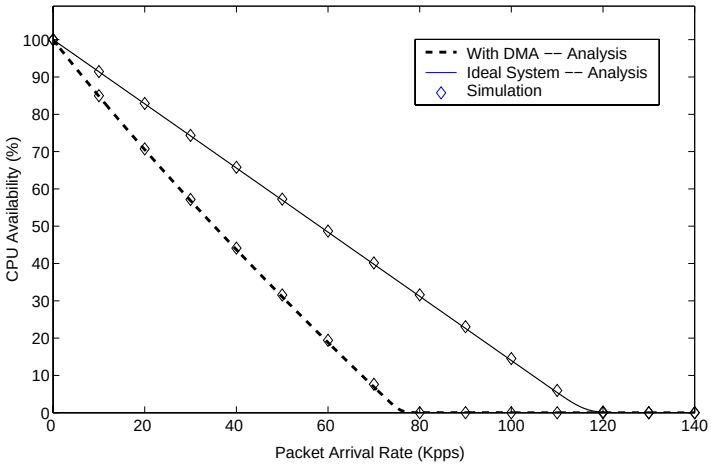

Figure 14. CPU Availability with DMA

When comparing PIO and DMA system throughput, one can conclude that DMA system throughput doesn't fall rapidly to zero, but it gradually decreases as the system load of packet arrivals increases. Another important observation can be made about the receive-livelock point in which the PIO system throughput reaches zero. From Figure 11, it is observed that the receive-livelock occurs at $\lambda = 1/c = 18{,}182$ pps. This observation is inline with our analytical solution. Based on equation (7), the receive-livelock point depends only the values of λ and c. It occurs precisely 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. On the other hand with DMA, the receive-livelock point does not occur unless λ becomes substantially large, as illustrated by equation (9).

The corresponding available CPU time percentage for PIO and DMA of Figure 11 and Figure 12 is plotted in Figure 13 and Figure 14, respectively. The results are shown with Poisson arrivals. It is depicted from both figures that the CPU availability reaches zero when the system becomes saturated. This happens exactly when $\rho = 1 \quad or \quad \lambda = \mu'$. At this point the CPU power is totally consumed by ISR handling and protocol processing. This is also the saturation point or the "cliff" point of system throughput. More importantly, this is the starvation or livelock point for user applications where the CPU available bandwidth becomes zero.

5. Discussion

We presented analytical models to capture the receive livelock phenomenon. As demonstrated, degraded throughput with little or no CPU power for applications can be encountered due to heavy network loads. Our contribution is primarily twofold. First, we studied receive livelock and provided insight into predicting such a system performance. Second, we identified critical operating points of performance when employing PIO and DMA. These operating points included stability and saturation conditions, throughput, and leftover CPU time. Based on our analysis and simulation work, the following design recommendations and guidelines must be considered in order to improve host performance:

Good Overload Performance is Critical. It is imperative to design a system with good overload conditions. A major contribution of our analytical work is identifying the overload condition. Maintaining good performance under overload conditions is critical. A system or a host under severe and heavy network traffic should sustain its throughput or capacity. Such throughput should not be degraded as the network load or traffic increases. We referred to the point, where the throughput starts being degraded, as the "cliff" or saturation point. It is also called the application starvation point. See Sections 3.3 and 3.4. Our analysis provided equations to predict, with adequate degree of accuracy, where this point occurs. This point was defined as λ_{cliff}, and given by equation (8) and equation (10) for PIO and DMA, respectively.

As a good design practice and in order to sustain the system throughput with no noticeable degradation at overload condition, precisely at the cliff point λ_{cliff}, the host should employ one of the techniques to mitigate

interrupts. One famous technique is disabling interrupts and starting polling [3,9]. Another technique is interrupt coalescing (IC) which is a feature provided by the NIC [1,22,23]. In polling, the OS periodically polls its host system memory (i.e., protocol processing buffer) to find packets to process. Typically there is a maximum number of packets to process in each poll in order to leave some CPU power for application processing. It is worth noting that unsuccessful polls can be encountered as packets are not guaranteed to be present at all times in the host memory, and thus CPU power is wasted. In IC, the NIC generates a single interrupt for multiple incoming packets. This is opposed to normal-interruption mode in which an interrupt is generated for every incoming packet. A key issue to note is that polling and IC decreases interrupt overhead on the expense of latency, as packets are not processed instantly but rather get queued to be polled or coalesced. At low load, polling and IC yield excessive latency. It is only practical to switch to polling or IC mode at overload condition, i.e., when mean load λ is close to λ_{cliff}. In this paper design recommendations are focused on switching between interrupts and polling, however, switching between normal interrupts and IC can be treated in a similar fashion whereby the coalescing parameter (that indicates how many packets to coalesce) can be tuned based on how close the mean load λ from λ_{cliff}.

Switching between interrupts and polling is proposed in [3,9]. However in [3,9], the overload condition was not identified accurately as is the case with our analytical study. In [9], the overload condition was based on the arrival rate and was chosen *arbitrarily* and has to be tuned manually. Also in [9], the overload condition was based on the host buffer occupancy and two levels of occupancy which were selected *arbitrarily*. Identifying properly where overload conditions occurs is important. In our analysis, the overload condition occurs at λ_{cliff}. Given the system parameters of interrupt overhead, protocol processing, and packet copy times in case of PIO, λ_{cliff} can be computed. We propose the use of two thresholds of operations: upper (U) and lower (L), where $U = \alpha\ \lambda_{cliff}$ and $L = \beta\ \lambda_{cliff}$. α and β are tunable and design parameters, and their value selection depends on how aggressive or relaxed the need of switching between interrupts and polling. The value selection also depends on the CPU availability percentage required to be reserved for application processing. Good design values for α and β can be 95% and 85%, respectively.

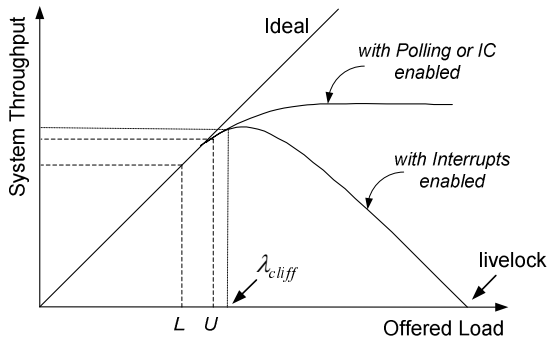

Figure 15. Critical design and operating points

As depicted in Figure 15, it is to be noted that as long as the host is operating in the region between *U* and *L* thresholds, no mode switching between interrupts and polling should take place. Using two thresholds is necessary in order to avoid possible significant overhead that may result from frequent fluctuation around one threshold point. When the arrival rate λ exceeds the upper threshold *U*, the host's OS must switch to polling mode. When the arrival rate λ becomes lower than the lower threshold *L*, the host's OS must switch to interrupt mode.

From implementation point of view, we propose two solutions to measure the overload condition and implement such a hybrid interrupt-polling scheme.

A) NIC-Side Solution. In this solution, the OS should initially set the values for *U* and *L* thresholds in the NIC. The NIC should be capable of computing λ by recording and measuring the inter-arrival times of incoming packets using exponential averaging method as reported in [24]. When $\lambda > U$, the NIC should notify the OS to disable interrupts and enable polling. When $\lambda < L$, the NIC should notify the OS to enable interrupts and disable polling.

B) OS-Side Solution. This solution should be employed when the NIC is not equipped with software to measure the inter-arrival times of incoming packets. In this solution the measurement of the overload condition is performed entirely by the OS. This solution requires more overhead on the part of the OS. There are three possible approaches to measure the overload condition by the OS:

- *CPU Utilization.* Monitoring the CPU utilization of the host in order to determine network overload condition is an invalid approach. This approach is stated here for the sake of discussion and coverage of all possible approaches. The CPU utilization can go high due to so many reasons other than interrupt handling and protocol processing. For example the CPU utilization can be high due to heavy CPU-bound processes or threads activities.
- *Host System Buffer Occupancy.* In this approach, the networking subsystem of the OS must periodically checks the status of kernel host buffer of where the incoming packets are being copied or DMA'd. This approach was proposed in [3]. If the buffer occupancy is at 75%, then the OS should disable interrupts and enable polling. Conversely if the buffer occupancy reaches a level of 25%, then the OS should enable interrupts and disable polling. In [3], the upper and lower levels of buffer occupancy were selected arbitrarily. According to [3], determining the proper upper and lower buffer occupancy is an arbitrary and in reality a non-trivial task. These levels vary significantly as they primarily depend on the size of the buffer being used. A keynote to remember is that at high load the buffer fills up very quickly. Also at low load with instantaneous burst of traffic the buffer also fills up very quickly. Therefore such a solution has potential shortcomings and is not recommended.
- *System Throughput.* We propose and recommend this approach when the NIC-side solution is not feasible. In this approach, the OS keeps track of the average system throughput of the packets that get processed and delivered to applications. This average system throughput was referred to analytically as γ by equation (1) with the corresponding μ' for PIO or DMA. Note that as illustrated by Figure 15, the point of overload condition occurs when $\gamma = \lambda_{cliff}$. Also note that *U* and *L* thresholds for arrival rate are the same *U* and *L* thresholds for system throughput. Hence, when $\gamma > U$, the OS must switch

to polling mode. When $\gamma < L$, the OS must switch to interrupts mode. This method is as accurate as that of the NIC-side solution, however this method requires more overhead on the part of the OS. The OS can use similar method, as that of the NIC, by recording and measuring the inter-arrival times of processed and delivered packets using exponential averaging method, discussed in [24].

Determining Maximum Throughput. Our analysis effort provided equations that can be used to easily and quickly predict the host performance and behavior. Given a system's design (DMA or PIO) and with certain known parameters of protocol processing time, application processing time and interrupt overhead, it would be useful to know how much traffic the system can process and how it would behave, even before building a prototype. As discussed in Section 3.3 and 3.4 and as shown in Figure 15, the maximum system capacity is basically λ_{cliff}, and is given by equation (8) and equation (10) for PIO and DMA, respectively. 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 tuning 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.

DMA vs. PIO Design Options. As demonstrated in the paper, receive livelock phenomenon is far worse in PIO than DMA. 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.

Latency and Queue Length. Our analysis is based on queueing theory and can be easily extended to predict other important performance indicators and to select proper design parameters. Some of the performance indicators may include latency, waiting time, and blocking probability. Important design parameters may include the proper size of the buffers for kernel's protocol processing as well as the application. [20] gives already derived and known formulas to compute many performance indicators and design parameters. The input parameters for these formulas depend primarily on λ and μ. Our analytic approach was based on finding the key input parameter μ' which is the mean protocol processing rate.

6. Conclusion

We developed analytical models to study the impact of interrupt overhead caused by Gigabit network traffic on system performance. Our analytical models were verified by simulation. Also reported experimental results show that our analytical models give a good approximation. Our work can be valuable in providing insight to understand and predict system behavior, as well as improving and maintaining good performance for the host. The paper identified critical design operation points for hosts with PIO and DMA. These operating points included: (1) stability and saturation conditions where system throughput starts to fall to zero, (2) application livelock where the application is starved as the CPU power is consumed by interrupt overhead and protocol processing., and (3) system livelock where the system throughput drops to zero. The paper also proposed

recommendations for implementing and improving host 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] 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*, 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. 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] A. Law and W. Kelton, *Simulation Modeling and Analysis*, McGraw-Hill, 2nd Edition, 1991.

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

[22] R. Prasad, M. Jain, and C. Dovrolis, "Effects of Interrupt Coalescence on Network Measurements," Proceedings of Passive and Active Measurement (PAM) Workshop, France, April 2004.

[23] M. Zec, M. Mikuc, and M. Zagar, "Estimating the Impact of Interrupt Coalescing Delays on Steady State TCP Throughput", Proceedings of the 10th SoftCOM, October 2002.

[24] A. Silberschatz, P. Galvin, and G. Gagne, "Operating System Concepts," John Wiley & Sons, Inc, 4th Edition, 2003.