Two Exact Analytical Models for Evaluating Performance of Gigabit Ethernet Hosts

Khaled Salah
Department of Information and Computer Science
King Fahd University of Petroleum and Minerals
Dhahran 31261, Saudi Arabia
Email: salah@kfupm.edu.sa

Abstract

Two exact analytical models are developed to study the impact of interrupt overhead on operating system performance of network hosts such as PC-based routers, servers, and end hosts when subjected to Gigabit network traffic. Under heavy network traffic, the system performance will be negatively affected due to interrupt overhead caused by incoming traffic. In particular, excessive latency and significant degradation in system throughput can be experienced. Also, user applications may livelock as the CPU power is mostly consumed by interrupt handling and protocol processing. In this paper, we present two exact analytical models to evaluate system performance. The system performance is studied in terms of throughput, latency, stability condition, CPU utilizations of interrupt handling and protocol processing, and CPU availability for user applications. 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. At an early stage, our analysis work can be valuable for engineering and designing certain system parameters. And at a later stage, the analysis can be used to aid in system calibration and diagnosis. In order to verify and validate our analytical models, we developed a discrete-event simulation and compared results to real lab experiment. Simulations and reported experimental results show that our analytical models are valid and give a good approximation.

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

1. Introduction

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

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

The receive livelock was established by experimental work on real systems in [2-4]. A number of solutions have been proposed in the literature [1,3,5-13] 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 throughput analysis was presented for interrupt-driven kernels when utilizing PIO and DMA in high-speed networks such as that of Gigabit Ethernet. In this paper, we present two exact analytical models that are based on queueing theory and Markov process. The models are exact because the give the same analytical equations and results for a number of important performance metrics. These performance metrics include system throughput, system latency, host saturation point and system stability condition, CPU utilizations of ISR handling and protocol processing, and CPU availability for other processing including user applications. As opposed to prototyping and simulation, these two 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. 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 two exact analytical models that capture the system behavior and study the performance of Gigabit Ethernet hosts. Section 4 verifies the two analytical model using a discrete-even simulation. Section 5 shows numerical examples to validate the analysis. 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.

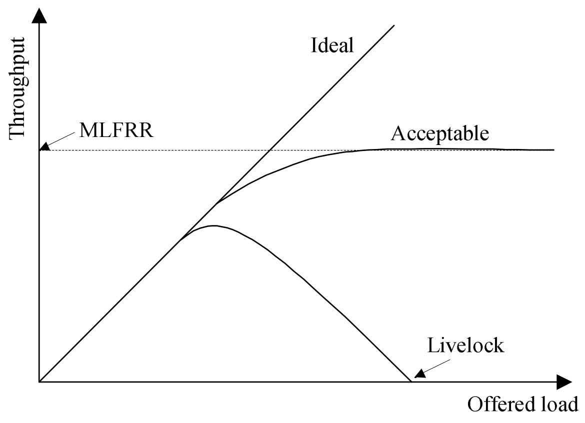


Figure 1. Receive livelock phenomenon

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.

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 two exact analytical models to examine the impact of interrupt overhead on OS 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. Note that $1/\mu$ is the average time the system takes to process the incoming packet and deliver it to the user application. This time includes primarily the network protocol stack processing carried out by the kernel, excluding any time disruption due to interrupt handling. Let $1/r$ be the mean interrupt handling time, which is basically the interrupt service routine time for handling incoming packets. $1/r$ includes basically the interrupt-context switching overhead and notifying the kernel to start the protocol processing for the received packet.

After the notification of the arrival of a new packet, the kernel will process the packet by first examining the type of frame being received and then invoking immediately the proper handling stack function or protocol, e.g.

Top

ARP, IP, TCP, UDP, etc. The packet will remain in the kernel or system memory until it is discarded or delivered to the user application. The network protocol processing for packets carried out by the kernel will continue as long as there are packets available in the system memory buffer. However, this protocol processing of packets can be interrupted by ISR executions as a result of new packet arrivals. This is so because packet processing by the kernel runs at a lower priority than the ISR.

There are two possible system delivery options of packet to 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,-8,13-16]. 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.

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. These times change due to various OS activities. For example ISR handling for incoming packets can be interrupted by other interrupts of higher priority, e.g. timer interrupts. Also, protocol processing can be interrupted by higher priority kernel tasks, e.g. scheduler. For our analysis, we assume these service times to be exponential. In Section 5, we demonstrate that this assumption gives an adequate approximation.

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

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

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 5, it turns out that our model with the above assumptions gives a good approximation to real experimental measurements. The impact of having constant network traffic instead of Poisson is currently being studied by the authors using simulation. Moreover, the impact of having variable-size packets, e.g. Jumbo frames, and other traffic distributions, e.g. bursty traffic [19], is also being studied by the author using simulations and results are expected to be reported in the near future.

3.2. DMA-Based Design

For our hosts, we assume that the NIC is equipped with DMA engines. However, a NIC adapter can be designed with a PIO-based option. A NIC adapter with PIO-based design can be an attractive option when considering factors such as cost, simplicity, and speed and efficiency in copying relatively small-size packets [20]. However, a major drawback for a PIO-based design is burdening the CPU with copying incoming packets from the NIC to kernel memory. In order to save CPU cycles consumed in copying packets, major network vendors equip high-speed NICs with DMA engines. These vendors include Intel, 3Com, HP, Alteon owned now by Nortel, Sundace, and NetGear. NICs are equipped with a receive Rx DMA engine and a

Top

transmit Tx DMA engine. A Rx DMA engine handles transparently the movement of packets from the NIC internal buffer to the host system memory. A Tx DMA engine handles transparently the movement of packets from the host memory to the NIC internal buffer. Both DMA engines operate in a bus-master fashion, i.e. the engines request access to the PCI bus instead of waiting to be polled. It is worth noting that the transfer rate of incoming traffic into the kernel memory across the PCI bus is not limited by the throughput of the DMA channel. These days a typical DMA engine can sustain over 1 Gbps of throughput for PCI 32/33 MHz bus and over 4 Gbps for PCI 64/66 MHz bus [21, 22].

It is important to note that the device driver for the network adapters is typically configured such that an interrupt is generated after the incoming packet has been completely DMA'd into the host system memory. In order to minimize the time for ISR execution, ISR handling mainly sets a software interrupt to trigger the protocol processing for the incoming packet. Please note in this situation if two or more packets arrive during an ISR handling, the ISR time for servicing all of these packets will be the ISR time for servicing a single packet, with no extra time introduced.

3.3. Modeling Interrupts is a Challenging and Difficult Task.

Modeling Interrupts is a challenging and difficult task. As noted earlier the ISR execution preempts protocol processing, and hence, 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. 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 two analytical models: one is based on an *M/M/1* queueing model and the other is a pure Markov process.

3.4. Analytical Model I

The model is based on first determining the CPU utilization for ISR handling, next finding the mean effective protocol processing rate, and then modeling the protocol processing as *M/M/1* queueing system with this mean effective rate. More details on *Analytical Model I* can be found in [17]. In [17], the system performance was only studied in terms of throughput. In this paper we extend the analysis work to examine more performance metrics, particularly system latency, saturation point, CPU utilizations of ISR handling and protocol processing, and CPU availability for user applications.

In order to find the CPU utilization percentage for ISR 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 $0 < n < \infty$ represents the state where the CPU is busy handling interrupts. n denotes the number of interrupts that are batched or masked off during ISR handling. Note that $n+1$ denotes the number of packet arrivals during ISR handling. Therefore, state (1,0) means there are no interrupts being masked off and the CPU is busy handling an ISR with one packet arrival. State (1,1) means that one interrupt has been masked off and the CPU is busy handling an ISR with two packet arrivals. Both of these packets will be serviced together with a mean rate r of servicing only one packet.

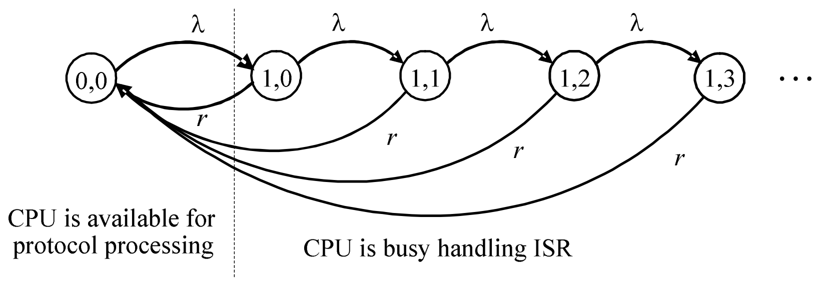


Figure 2. Markov state transition diagram for modeling CPU usage with DMA

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

Therefore, the CPU utilization for ISR handling is $\dfrac{\lambda}{\lambda + r}$. The mean effective service rate μ' for protocol processing can be computed in terms of CPU percentage availability for protocol processing. The mean effective service rate can be expressed as

$$\mu' = \mu \times \text{(\% CPU availability for protocol processing)},$$

$$\mu' = \mu \cdot \frac{r}{\lambda + r}. \tag{1}$$

The term $\dfrac{r}{\lambda + r}$ is the percentage of CPU bandwidth available for protocol processing, and is equal to $1 - \dfrac{\lambda}{\lambda + r}$.

CPU Availability. For such a model, the percentage of CPU power available for other processing, including user applications, is basically the probability when there is no ISR handling and there are no packets being processed by the protocol stack. It is to be noted from equation (1) that the mean effective service time $1/\mu'$ is exponential. Therefore, the protocol processing can be modeled as *M/M/1* queue with a mean service rate of μ'. Hence, the CPU availability for other processing can be expressed as

$$V = \left(\frac{r}{\lambda + r}\right) \cdot p_0$$

where p_0 is the probability of not queueing, i.e. finding zero packets, in the *M/M/1* queueing system of the kernel's protocol processing. $p_0 = 1 - \rho_{IP}$, where $\rho_{IP} = \left(\frac{\lambda}{\mu'}\right)$. Note that ρ_{IP} is the network load, or traffic intensity, being encountered due to kernel's protocol processing.

Hence, V can be further simplified to

$$V = \left(\frac{r}{\lambda + r}\right) - \frac{\lambda}{\mu}. \tag{2}$$

CPU Utilization. The CPU utilization ρ which includes ISR handling and protocol processing can be expressed as

$$\rho = 1 - V = \left(\frac{\lambda}{\lambda + r}\right) + \frac{\lambda}{\mu}. \tag{3}$$

Note this CPU utilization gives a sensible result. The utilization is basically the sum of the CPU utilization for ISR handling $\frac{\lambda}{\lambda + r}$ and for protocol processing $\frac{\lambda}{\mu}$. We can also think of the CPU utilization for protocol processing as the probability of no ISR handling and having one or more packets being processed by the protocol stack, i.e. $\frac{\lambda}{\lambda + r} \cdot \left(1 - \frac{\lambda}{\mu'}\right)$, which can be simplified nicely to $\frac{\lambda}{\mu}$. In addition, it is to be noted that CPU availability V derived by equation(2) gives a sensible results. It shows that one can obtain V simply by subtracting the CPU utilization of for protocol processing $\frac{\lambda}{\mu}$ from the CPU availability for protocol processing $\frac{r}{\lambda + r}$.

Saturation Point. A critical operating point for the system is computing the saturation point. It is the point at which the system can not keep up with the offered network load, i.e. $\rho = 1$. This is also referred to the "cliff" point of system throughput, i.e. $\lambda = \mu'$. It is where the throughput starts falling as the network load increases. Also the system will become unstable causing dropping of packets, excessive latencies and timeouts. In addition, the user applications will livelock at this point as the CPU power is at 100%, i.e. $V = 0$. The CPU power is being consumed by ISR handling and protocol processing. The saturation condition can be expressed as

$$\rho = 1 \quad or \quad V = 0 \quad or \quad \lambda = \mu'. \tag{4}$$

Solving for λ can be done two ways which give the same outcome. One way is to substitute equation (2) for $\rho = 1$. Second is to substitute equation (1) or $\lambda = \mu'$. The saturation point can be derived and solved for λ as follows:

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

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 system will be stable whenever

$$\lambda = \frac{r}{2}\left(\sqrt{1 + 4\frac{\mu}{r}} - 1\right). \tag{5}$$

Mean System Throughput. The mean system throughput γ is basically the departure rate due to protocol processing, and it can be expressed as

$$\gamma = \mu'(1 - p_0) = \mu'(1 - 1 + \frac{\lambda}{\mu'}) = \lambda. \tag{6}$$

Mean System Latency. The mean system latency 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

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

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

$E_{ISR}(r)$ 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 as

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

With further simplification,

Top

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

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

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

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

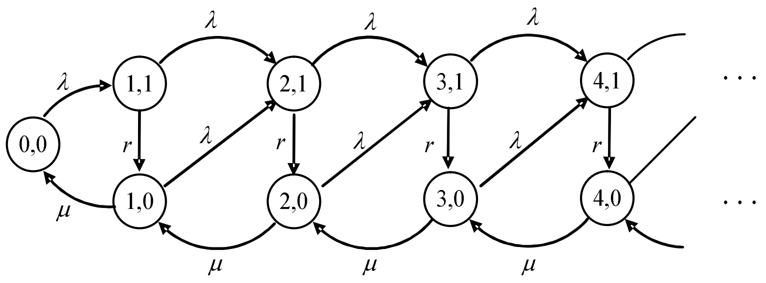
$$E_{IP}(r) = \frac{E_{IP}(n)}{\lambda} = \left(\frac{\rho_{IP}}{1-\rho_{IP}}\right)\frac{1}{\lambda} = \frac{1}{\mu'-\lambda}.$$

Therefore, the mean system delay, according to Jackson's theorem, is

$$E(r) = \frac{1}{r} + \frac{1}{\mu'-\lambda}. \qquad (8)$$

3.5. Analytical Model II

This model captures the behavior of the interrupt-driven system using only a Markov process with no *M/M/1* queueing. The interrupt-driven system with DMA design option can be modeled as a pure Markov chain with a state space $S = \{ (n,m), 0 \le n \le \infty, m \in \{0,1\} \}$, where n denotes the number of packets in the buffer and m denotes the type of activity the CPU is performing. State (0,0) represents the state where the CPU is idle. States $(n,1)$ represent the states where the CPU is busy handling interrupts. States $(n,0)$ represent the states where the CPU is busy processing protocol. The rate transition diagram is shown in Figure 3.



$$\begin{tikzcd}
 & 1,1 \arrow[d, "r"] \arrow[r, bend left, "\lambda"] & 2,1 \arrow[d, "r"] \arrow[r, bend left, "\lambda"] & 3,1 \arrow[d, "r"] \arrow[r, bend left, "\lambda"] & 4,1 \arrow[d, "r"] \arrow[r, bend left] & \cdots \\
0,0 \arrow[ur, bend left, "\lambda"] & 1,0 \arrow[l, bend left, "\mu"] \arrow[ur, "\lambda"] & 2,0 \arrow[l, bend left, "\mu"] \arrow[ur, "\lambda"] & 3,0 \arrow[l, bend left, "\mu"] \arrow[ur, "\lambda"] & 4,0 \arrow[l, bend left, "\mu"] \arrow[ur] & \cdots \arrow[l, bend left]
\end{tikzcd}$$

Figure 3. Markov state transition diagram for interrupt-driven system with DMA

Let $p_{n,m}$ be the steady-state probability at state (n,m). A system of difference equations can be derived for the stationary probabilities as follows:

$$
\begin{aligned}
&0 = -\lambda p_{0,0} + \mu p_{1,0}, && \\
&0 = -(\lambda + r) p_{1,1} + \lambda p_{0,0}, && \\
&0 = -(\lambda+\mu) p_{n,0} + r p_{n,1} + \mu p_{n+1,0} && \text{for } n \ge 1, \\
&0 = -(\lambda + r) p_{n,1} + \lambda p_{n-1,0} + \lambda p_{n-1,1} && \text{for } n \ge 2.
\end{aligned} \tag{9}
$$

The first two equations constitute the initial values. The last two equations constitute the system of difference equations. In order to solve this system of equations, we need to re-arrange them as follows:

$$
\begin{aligned}
p_{n+1,0} &= \frac{\lambda+\mu}{\mu} p_{n,0} - \frac{r}{\mu} p_{n,1} && n \ge 1, \\
p_{n+1,1} &= \frac{\lambda}{\lambda+r} p_{n,0} + \frac{\lambda}{\lambda+r} p_{n,1} && n \ge 1.
\end{aligned}
$$

These equations can be written in the vector form as

$$p(n+1) = A\,p(n),$$

where

$$A = \begin{bmatrix} \dfrac{\lambda+\mu}{\mu} & -\dfrac{r}{\mu} \\ \dfrac{\lambda}{\lambda+r} & \dfrac{\lambda}{\lambda+r} \end{bmatrix},$$

$$p(n) = \begin{bmatrix} p_{n,0} \\ p_{n,1} \end{bmatrix}, \text{ and } p(n+1) = \begin{bmatrix} p_{n+1,0} \\ p_{n+1,1} \end{bmatrix}.$$

Therefore, our equations have been nicely converted to a system of first order difference equation, in which we can apply Putzer algorithm to obtain the solution [24].

Before we proceed further, let us denote $\alpha = \lambda/\mu$, and $\beta = \lambda/(\lambda + r)$. Then, matrix A can be rewritten as

$$A = \begin{bmatrix} \alpha + 1 & -\alpha(1-\beta)/\beta \\ \beta & \beta \end{bmatrix}.$$

The eigenvalues of matrix A can be obtained by solving the characteristic equation $\det(A - zI) = 0$ where z is the eigenvalue, and I is the identity matrix. Now

$$\det(A - zI) = \det\begin{bmatrix} \alpha + 1 - z & -\alpha(1-\beta)/\beta \\ \beta & \beta - z \end{bmatrix} = (1-z)(z-\alpha-\beta) = 0.$$

Hence, the eigenvalues of matrix A are $z_1 = 1$ and $z_2 = \alpha + \beta$.

So, according to Putzer Algorithm,

$$M(0) = I, \quad \text{and} \quad M(1) = A - z_1 I = \begin{bmatrix} \alpha & -\alpha(1-\beta)/\beta \\ \beta & \beta - 1 \end{bmatrix}.$$

Top

Then,

$$u_1(n) = 1^n = 1,$$

and

$$u_2(n) = \sum_{i=0}^{n-1} (\alpha+\beta)^{n-1-i}(1^i) = \frac{1-(\alpha+\beta)^n}{1-(\alpha+\beta)}.$$

Finally, we have

$$A^n = u_1(n) \times M(0) \quad + \quad u_2(n) \times M(1)$$
$$= \begin{bmatrix} \dfrac{1-\beta-\alpha(\alpha+\beta)^n}{1-(\alpha+\beta)} & \dfrac{\alpha(1-\beta)(1-(\alpha+\beta)^n)}{\beta(1-(\alpha+\beta))} \\ \dfrac{\beta(1-(\alpha+\beta)^n)}{1-(\alpha+\beta)} & \dfrac{-\alpha+(1-\beta)(\alpha+\beta)^n}{1-(\alpha+\beta)} \end{bmatrix}.$$

The solution of the difference equation is given by

$$p(n+1) = A^n p(1) = \begin{bmatrix} \dfrac{1-\beta-\alpha(\alpha+\beta)^n}{1-(\alpha+\beta)} & \dfrac{\alpha(1-\beta)(1-(\alpha+\beta)^n)}{\beta(1-(\alpha+\beta))} \\ \dfrac{\beta(1-(\alpha+\beta)^n)}{1-(\alpha+\beta)} & \dfrac{-\alpha+(1-\beta)(\alpha+\beta)^n}{1-(\alpha+\beta)} \end{bmatrix} \times \begin{bmatrix} \alpha\, p_{0,0} \\ \beta\, p_{0,0} \end{bmatrix}$$
$$= \begin{bmatrix} \dfrac{1-\beta-\alpha(\alpha+\beta)^n}{1-(\alpha+\beta)} \times \alpha\, p_{0,0} + \dfrac{\alpha(1-\beta)(1-(\alpha+\beta)^n)}{\beta(1-(\alpha+\beta))} \times \beta\, p_{0,0} \\ \dfrac{\beta(1-(\alpha+\beta)^n)}{1-(\alpha+\beta)} \times \alpha\, p_{0,0} + \dfrac{-\alpha+(1-\beta)(\alpha+\beta)^n}{1-(\alpha+\beta)} \times \beta\, p_{0,0} \end{bmatrix}$$

The solution can be nicely simplified to

$$\left.\begin{aligned} p_{n,0} &= \alpha\, p_{0,0}(\alpha+\beta)^{n-1} \\ p_{n,1} &= \beta\, p_{0,0}(\alpha+\beta)^{n-1} \end{aligned}\right\} \quad n \ge 1 \qquad (10)$$

To get $p_{0,0}$, we utilize the fact that all probabilities must sum up to 1. Hence,

$$\sum_{n=1}^{\infty} p_{n,0} + \sum_{n=1}^{\infty} p_{n,1} + p_{0,0} = 1,$$

$$p_{0,0}\,\alpha \sum_{n=1}^{\infty} (\alpha+\beta)^{n-1} + p_{0,0}\,\beta \sum_{n=1}^{\infty} (\alpha+\beta)^{n-1} + p_{0,0} = 1.$$

Therefore

$$p_{0,0} = \left[1 + (\alpha+\beta)\sum_{n=1}^{\infty} (\alpha+\beta)^{n-1}\right]^{-1}.$$

Now $\sum_{n=1}^{\infty} (\alpha+\beta)^{n-1}$ is geometric series and converges if and only if $(\alpha+\beta) < 1$. Thus for the existence of a steady-state solution, $\rho = (\alpha+\beta)$ must be less than 1. Then, we have

$$p_{0,0} = \left[1 + \frac{\alpha+\beta}{1-(\alpha+\beta)}\right]^{-1} = 1-(\alpha+\beta) = 1-\rho .$$

where $\rho = (\alpha+\beta)$, or equivalently, $\rho = \lambda/\mu + \lambda/(\lambda+r)$.

Thus the full steady-state solution for our system is the geometric probability functions

$$\begin{aligned} &p_{0,0} = 1-\rho \\ &\left.\begin{aligned} p_{n,0} &= \alpha(1-\rho)\rho^{n-1} \\ p_{n,1} &= \beta(1-\rho)\rho^{n-1} \end{aligned}\right\} \quad n \geq 1 \end{aligned} \tag{11}$$

where $\rho = \alpha+\beta$, $\alpha = \lambda/\mu$, and $\beta = \lambda/(\lambda+r)$.

CPU Utilization and Availability. Using the pure Markovian model, the CPU utilization for ISR handling can be derived as

$$\sum_{n=1}^{\infty} p_{n,1} = \sum_{n=1}^{\infty} \beta(1-\rho)\rho^{n-1} = \beta(1-\rho)\sum_{n=1}^{\infty}\rho^{n-1} = \lambda/(\lambda+r) .$$

Similarly, the CPU utilization for protocol processing can be derived as

$$\sum_{n=1}^{\infty} p_{n,0} = \sum_{n=1}^{\infty} \alpha(1-\rho)\rho^{n-1} = \alpha(1-\rho)\sum_{n=1}^{\infty}\rho^{n-1} = \lambda/\mu .$$

Hence, the CPU utilization ρ due to both ISR handling and protocol processing is $\lambda/\mu + \lambda/(\lambda+r)$. The CPU availability V for other processes, including user applications, is basically $1-\rho$. The CPU availability is also given in equation (11) by $p_{0,0}$. Note that V given here for the pure Markovian process matches exactly V given in equation (2) for *Analytical Model I.*

Saturation Point. The saturation or the cliff point using the pure Markovian model occurs when

$$\rho = 1 \quad or \quad \lambda/\mu + \lambda/(\lambda+r) = 1.$$

The saturation point can be solved for λ and can be expressed exactly as in *Analytical Model I* given by equation (4).

Mean System Throughput. The mean system throughput, γ, for the pure Markovian model is the rate at which packets are successfully being processed by the kernel's protocol stack. According to [25], γ can be expressed as $\mu\sum_{n=1}^{\infty} p_{n,0}$. Therefore, γ can be derived as follows

$$\gamma = \mu\sum_{n=1}^{\infty} p_{n,0} = \mu\sum_{n=1}^{\infty}\alpha(1-\rho)\rho^{n-1} = \mu\alpha(1-\rho)\sum_{n=1}^{\infty}\rho^{n-1} = \mu\alpha(1-\rho)\times\frac{1}{1-\rho} = \lambda .$$

This equation and equation (6) of *Analytical Model I* are exact and are mathematically equivalent.

Mean System Latency. The mean system latency, *E(r)*, for the pure Markovian model can be computed as follows

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

where $E(n)$ is the expected number of packets in the system and can be expressed as

$$E(n) = \sum_{n=1}^{\infty} n(p_{n,0} + p_{n,1}) = \sum_{n=1}^{\infty} n \times \left[\alpha(1-\rho)\rho^{n-1} + \beta(1-\rho)\rho^{n-1}\right]$$

$$= (\alpha+\beta)(1-\rho)\sum_{n=1}^{\infty} n\rho^{n-1} = \rho(1-\rho)\left(\frac{1}{1-\rho}\right)^2 = \frac{\rho}{1-\rho}$$

Therefore,

$$E(r) = \left(\frac{\rho}{1-\rho}\right)\frac{1}{\lambda}.$$

Note in this case, his equation is not mathematically equivalent to equation (8) of *Analytical Model I.* As it will be demonstrated in Section 5 when giving numerical examples, both equations yield very close matching results. In fact at light load, the equations yield exactly matching results. The reason that these equations are not mathematically equivalent can be linked to the approximation method utilized in Jackson's theorem. Another reason can be contributed to the fact of having two series summations in deriving $E(n)$ in the pure Markovian process, as opposed to only one series summation in deriving $E(n)$ in the *M/M/1/B* queueing model. The derivation of $E(n)$ for an *M/M/1/B* queueing model is shown in [18].

3.6. Comparison between the Two Models

From Section 3.3 and Section 3.4, it can be concluded that the two analytical models give exactly the same equations and results for all system performance metrics, except for system latency. The exact mathematical solutions were given for system performance metrics which include system throughput, host saturation point and system stability condition, CPU utilizations of ISR handling and protocol processing, and CPU availability for other processing including user applications application. For system latency, the equations given by the two models are not mathematically equivalent. However and as will be demonstrated in Section 5 and Figure 5c, the equations give very close matching results. In fact, an exact matching of results exists at light system load. Therefore, it would be appropriate to conclude that the two analytical models are exact in general.

It should be noted that conducting analysis using *Analytical Model I* is easier and more convenient than that of *Analytical Model II.* Once the CPU utilization is determined for ISR handling, the performance metrics can be directly computed by applying known and already derived equations for *M/M/1* queueing system [18]. Such a technique is currently being utilized to examine and compare the performance of different proposed schemes for minimizing and eliminating the interrupt overhead caused by heavy network loads. Conducting analysis using *Analytical Model II* for such proposed methods will give intractable mathematical solution.

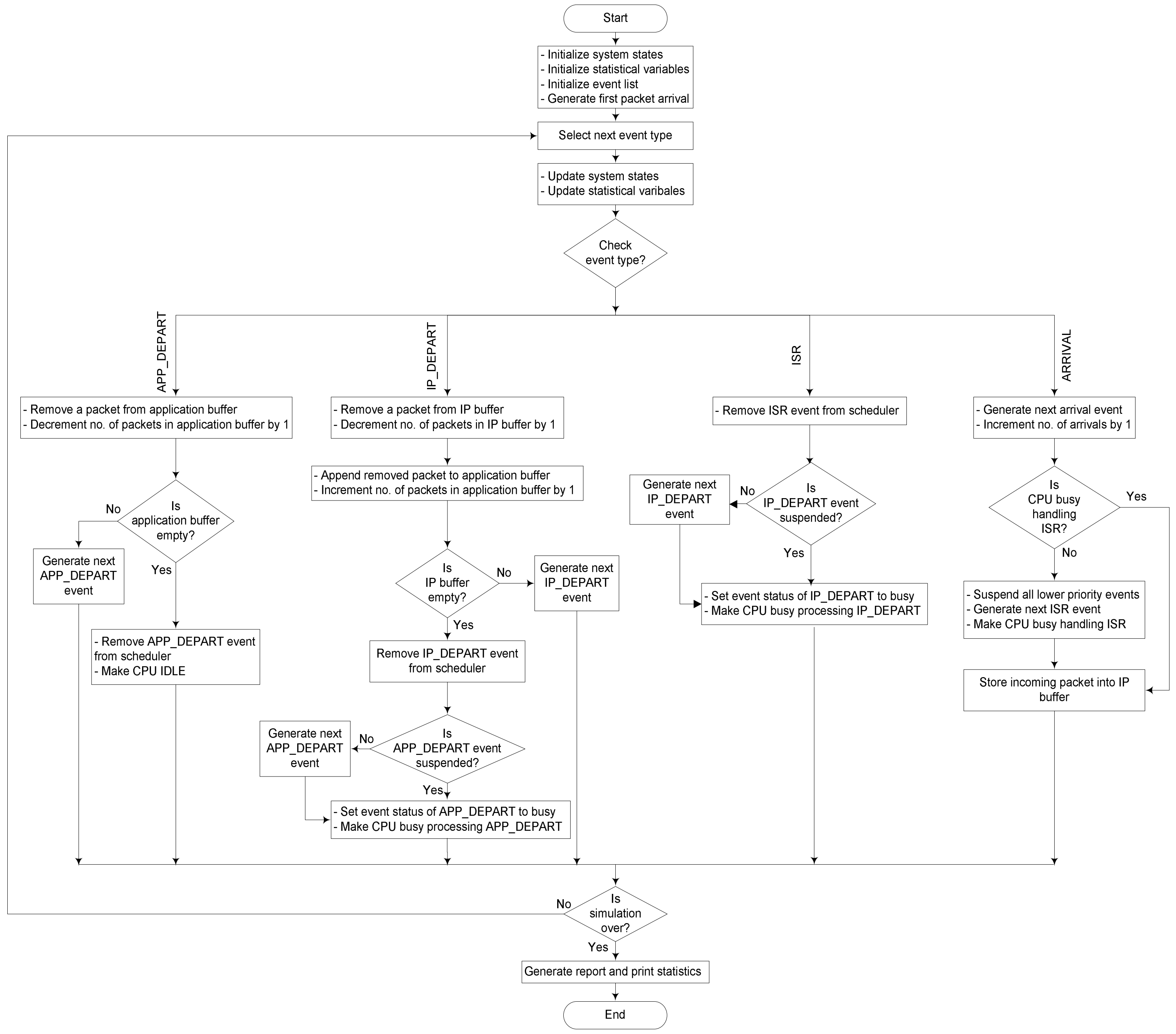
4. Simulation

In order to verify the analytical models, a discrete-event simulation was developed and written in C language. The simulation follows closely and carefully the guidelines given in [23]. Figure 4 shows the simulation flowchart for interrupt-driven kernel using DMA. Table 1 describes the events. Except for the ARRIVAL event, each event has a time, status, and priority. An event status can be IDLE, BUSY, or SUSPEDNDED. IDLE indicates the event has not been selected by the scheduler, i.e, not being served by the CPU. BUSY indicates the event is being served by the CPU. SUSPENDED indicates the event was BUSY but got preempted by a higher priority event. Only IP_DEPART and APP_DEPART events can have SUSPENDED status. The selection of the next event by the scheduler is based on the event's time, status, and priority. Whenever a SUSPENDED event is selected again to run (i.e., resumed running), its finish time will incur the service times of all higher priority events which occurred between its suspension and its resumption. The simulation has two buffers implemented as FIFO queues: IP and application. These queues hold the value of the arrival time of each packet. The simulation run ends when the total number of events reaches five millions.

Table 1. Simulation events

Event	**Description**
ARRIVAL	Occurs when a new packet arrives into the NIC.
ISR	Occurs when a packet received successfully by the NIC and the CPU is not busy handling an ISR.
IP_DEPART	Occurs when the CPU is not busy handling ISR 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 busy handling ISR, the IP buffer is empty, and the application buffer has packets. APP_DEPART indicates the completion of handling a packet by the user application.

Top



Start
- Initialize system states
- Initialize statistical variables
- Initialize event list
- Generate first packet arrival
Select next event type
- Update system states
- Update statistical varibales
Check event type?
APP_DEPART
IP_DEPART
ISR
ARRIVAL
- Remove a packet from application buffer
- Decrement no. of packets in application buffer by 1
Is application buffer empty?
No
Yes
Generate next APP_DEPART event
- Remove APP_DEPART event from scheduler
- Make CPU IDLE
- Remove a packet from IP buffer
- Decrement no. of packets in IP buffer by 1
- Append removed packet to application buffer
- Increment no. of packets in application buffer by 1
Is IP buffer empty?
No
Yes
Generate next IP_DEPART event
Remove IP_DEPART event from scheduler
Is APP_DEPART event suspended?
No
Yes
Generate next APP_DEPART event
- Set event status of APP_DEPART to busy
- Make CPU busy processing APP_DEPART
- Remove ISR event from scheduler
Is IP_DEPART event suspended?
No
Yes
Generate next IP_DEPART event
- Set event status of IP_DEPART to busy
- Make CPU busy processing IP_DEPART
- Generate next arrival event
- Increment no. of arrivals by 1
Is CPU busy handling ISR?
Yes
No
- Suspend all lower priority events
- Generate next ISR event
- Make CPU busy handling ISR
Store incoming packet into IP buffer
Is simulation over?
No
Yes
Generate report and print statistics
End

Figure 4. Simulation flowchart for interrupt-driven kernel with DMA

5. Numerical Examples

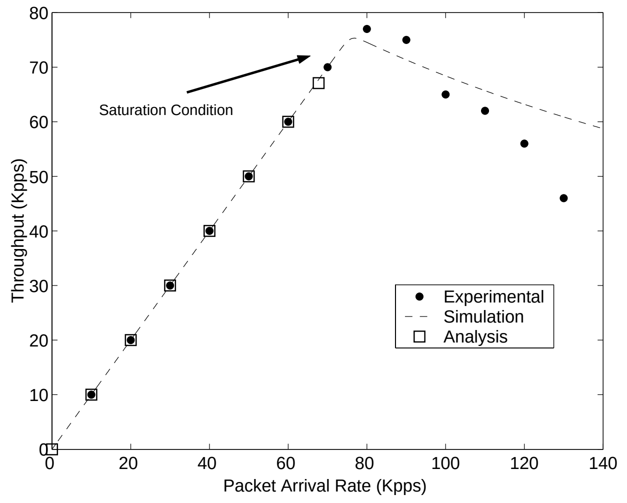
In this section, we report and compare results of analysis and simulation. Numerical results are given for the mean system throughput, latency, and CPU utilization. For validation, we compare our analysis and simulation results of system throughput to the DMA experimental results reported in [4].

In [4], the lab 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. 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.

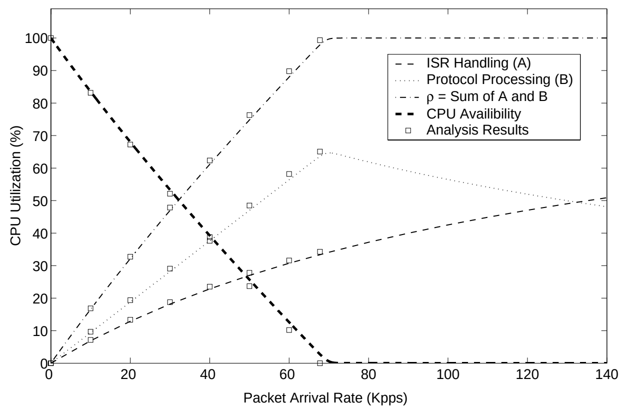
Figure 5a, Figure 5b, and Figure 5c plot the mean system throughput, CPU utilization, and mean system latency, respectively, as a function of packet arrival rate. Analysis results are only plotted when the system is stable, i.e. up to the saturation point. As for validation, we compare the experimental results for system throughput to those of analysis. CPU utilization and availability as well as system latency using real experiments were not measured in [4]. From the figure, it is clear that the discrete-event simulation results are very much in line with those of analysis. It is also depicted that the analysis results give an adequate approximation to real experimental measurements. For throughputs, Figure 5a depicts that the analysis and simulation results match exactly those of experimental at light load, i.e. in the stable region. At high load, there is a slight difference between simulation and experimental results. This difference can be contributed to the arrival characteristics of incoming packets. Our analysis assumed a Poisson arrival; however, as stated earlier, the inter-arrival times of the packets are not purely exponential. In the experiment, the packets were generated back-to-back by another host with almost a constant rate. Another factor that may contribute to the difference of experimental and simulation results for throughputs can be linked to the fact that the distribution of the service times for protocol processing and ISR handling are not precisely exponential. In our analysis, we assumed exponential distribution for both of the theses service times. Other factors can also be related to internal caching, system background processes, user-to-kernel mode switching, and measurement overhead.

When examining the mean system throughput of Figure 5a and the corresponding CPU utilization and mean system latency of Figure 5b and Figure 5c, it can be noted that the saturation point for the system occurs at $\lambda = 67{,}750$ pps. At this point, the corresponding CPU utilizations for both ISR handling and protocol processing is at 100%, with CPU availability of zero. Therefore, user applications will starve and livelock at this point. In addition, the mean system delay will shoot rapidly to infinity, causing excessive latency and timeouts. It can also be observed from Figure 5b that if the incoming traffic increases beyond the staturation point, as captured by the simulation, that the CPU utilizations for ISR handling keeps increasing and IP processing keeps decreasing, as expected. This causes more instability for the system as throughput worsens and packets start being dropped, causing a total collapse in both the system and user-perceived performance.

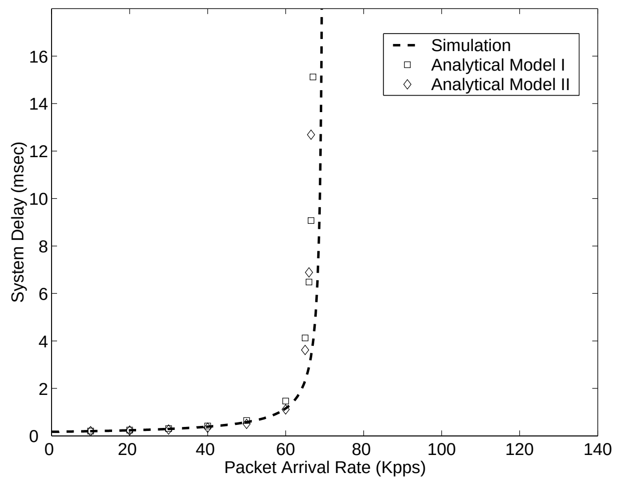
Figure 5c compares the results of *Analytical Model I* with those of *Analytical Model II*. It is depicted from the figure that both models give very close matching results. The results are exact in the light load and are very closely matching around the saturation point of the system. The results of both analytical models for other performance metrics are not plotted because they are mathematically equivalent as discussed in Section 3.5.



(a)



(b)



(c)

Figure 5. Mean system throughput , CPU utilization, and mean system latency

Top

6. Conclusion

We developed two analytical models to study the impact of interrupt overhead caused by Gigabit Ethernet network traffic on OS performance. The two models are exact in general. The analytical techniques employed for both models can be utilized to model and analyze other similar systems. In fact, *Analytical Model I* is currently being utilized by the author to evaluate the performance of the proposed schemes for resolving receive livelock and eliminating interrupt overhead. As demonstrated in the paper, *Analytical Model I* is more convenient and can yield more tractable mathematical solution than *Analytical Model II. Analytical Model I* is based on *M/M/1* queueing system and hence known equations can be directly applied to compute different performance metrics. Using these two exact models, we were able to conduct a throughput-delay analysis to evaluate the system performance of Gigabit Ethernet hosts when subjected to light and heavy network loads. We also investigated the system saturation point and stability condition, CPU utilizations of ISR handling and protocol processing, and CPU availability for other processing including user applications. As noted, degraded throughput, excessive latency, and starvation of user applications can be encountered due to heavy network loads. If the system is poorly designed, these penalties are quite high and can hurt the overall performance. Our analysis effort provided equations that can be used to easily and quickly predict the system performance and behavior when engineering and designing network adapters, device drivers, and OS network and communication software. Given a worst-case network load, acceptable performance levels for throughput, delay, and CPU availability can be reached by choosing the proper system parameters for protocol processing and ISR times. An acceptable performance level varies from one system requirement to another and depends on user-application requirements and the worst tolerable system throughput and latency. The two exact analytical models were verified by simulation. Also reported experimental results show that our analytical models give a good approximation. The impact of generating variable-size packets instead of fixed-size and bursty traffic instead of Poisson is being studied 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, the author is currently studying and evaluating the performance of the different proposed schemes for minimizing and eliminating the interrupt overhead caused by heavy network loads.

Acknowledgement

The author wishes to thank Mr. Khalid El-Badawi for his valuable feedback and input. The author also acknowledges 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. Badawi, "Evaluating System Performance in Gigabit Networks", The 28th IEEE Local Computer Networks (LCN), Bonn/Königswinter, Germany, October 20-24, 2003, pp. 498-505

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

[24] S. N. Elaydi, S. N., *An Introduction to Difference Equations*, Springer-Verlag, 1996, pg 113.

[25] B. Trivedi, *Queueing Networks and Markov Chains*, John Wiley & Sons, 1998.

Top