Two Analytical Models for Evaluating Performance of Gigabit Ethernet Hosts with Finite Buffer

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 analytical models are developed to study the impact of interrupt overhead on operating system performance of network hosts with limited-size or finite buffer. Under heavy network traffic such as that of Gigabit Ethernet, the system performance will be negatively affected due to interrupt overhead caused by incoming traffic. In particular, packet loss, 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 and compare two analytical models that capture host behavior and evaluate its performance. The first model is based on Markov processes and queueing theory, while the second, which is more accurate but more complex, is a pure Markov process. The models yield equations for a number of important system performance metrics. These performance metrics include throughput, latency, packet loss, stability condition, CPU utilizations of interrupt handling and protocol processing, and CPU availability for user applications. Both models yield closed-form solutions and equations that are either mathematically equivalent or very closely matching. Our 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. More importantly, our analytical work can also be valuable in improving host performance. The paper gives guidelines and recommendations to address design and implementation issues. Simulation and reported experimental results show that our analytical models are valid and give a good approximation.

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

1. Introduction

Interrupt overhead of Gigabit network devices can have a significant negative impact on system performance. Traditional operating systems were designed to handle network devices that interrupt on a rate of around 1000 packets per second, as is the case for 10Mbps Ethernet. 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. For 100Mbps Ethernet, the interrupt rate increases to about 8000 interrupts per second using the

[**] Correspondence to: Prof. K. Salah, PO Box 5066, ICS Department, KFUPM, Dhahran 31261, Saudi Arabia, phone: +96638604493 fax: +96638602174

standard maximum 1500 byte packets. However for Gigabit Ethernet, the interrupt rate for the maximum sized-packet of 1500 bytes increases to 80,000 interrupts per second. Of course with 10 Gigabit Ethernet and considering smaller packets, the problem is much worse.

In Gigabit networks, the packet arrival rate surpasses the system packet processing rate which includes network protocol stack processing and interrupt handling. With Gigabit Ethernet and a rate of 80,000 interrupts per second for a minimum sized packet of 512 bytes, the CPU must handle an interrupt in less than 4 μs in order to keep up with such a rate. According to [1], a *null* system call (not an interrupt) on a typical 666 MHz Intel Pentium III takes on the order of 10 μs. Also, a typical latency for handling interrupt due to a packet arrival in Linux is in the order of 50 μs. It is important to notice that with the presence of more powerful multi gigahertz processors these days, it is expected the interrupt cost will not be reduced linearly by the speed frequency of the processor, as I/O and memory speed limits dominate [2]. In [2] it was concluded that the performance of 2.4GHz processor only scales to approximately 60% of the performance of an 800MHz processor.

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* [3]. 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 [3-5]. A number of solutions have been proposed in the literature [4,6-15] 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 [10,14], a simple calculation of the interrupt overhead was presented. In [10], 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 [14], 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 [16], 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 and compare two analytical models to capture host behavior and evaluate its performance. We consider hosts with limited-size or finite buffer. For the most part both models give mathematically-equivalent closed-form solutions for a number of important system performance metrics. These performance metrics include system throughput, system latency, packet loss, 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 paper also gives guidelines and recommendations to improve overall system performance.

The rest of the paper is organized as follows. Section 2 describes the receive livelock phenomenon reported in literature. Section 3 presents two mathematically-equivalent analytical models that capture the system behavior and study the performance of Gigabit Ethernet hosts. Section 4 shows numerical examples to compare and validate the analysis. Section 5 gives some guidelines and recommendations to address design and implementation issues. 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 [3,4], 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) [3]. Such rate is an acceptable rate and is relatively flat after that.

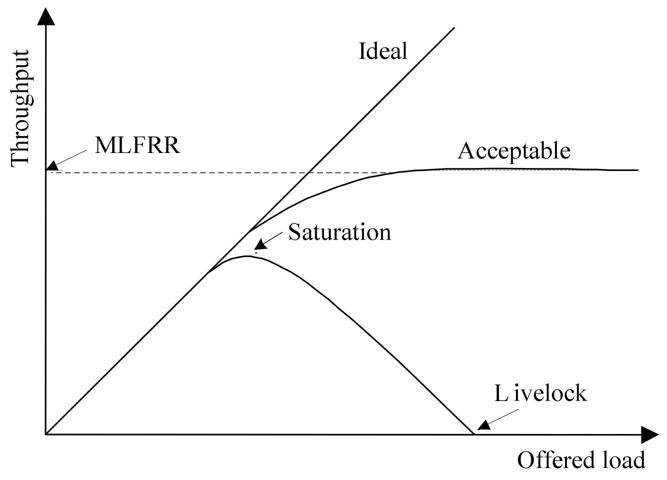


Figure 1. Receive livelock phenomenon

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 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. $1/r$ basically includes the interrupt-context switching overhead as well as the ISR handling. The main function of ISR handling is to notify the kernel to start protocol processing of the received packet. In [10,15], ISR handling included flushing DMA'd incoming packets from kernel's host memory to protocol incoming buffer. Keeping $1/r$ to minimum execution is highly desirable. Hence, flushing of incoming packets is highly recommend to be performed outside of the ISR and to be combined with protocol processing.

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, 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 [7-9,14,17-19]. 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 4, we demonstrate that this assumption gives an adequate approximation.

ii) The network traffic follows a Poisson process, i.e., the packet interarrival times are exponentially distributed. In many situations, assuming Poisson arrivals is adequate. In [20], it was concluded that modeling the voice traffic as Poisson gives adequate approximation, especially if the voice traffic is high.

iii) The packet sizes are fixed. This assumption is true for Constant Bit Rate (CBR) traffic such as uncompressed interactive audio and video conferencing.

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 [21,22], are currently 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 [23]. 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 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 [24, 25].

It is important to note that the NIC 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/G/1/B* 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/G/1/B* queueing system with this mean effective rate. *M/G/1/B* queueing model is chosen as opposed to *M/G/1* for two important reasons. First, in *M/G/1/B*, the arrival rate go beyond the service rate, i.e., $\lambda > \mu$. This assumption is a must for Gigabit environment where under heavy load λ can be very high compared to μ. Second, hosts practically and realistically has a finite amount of buffer space reserved for protocol processing. More details on *Analytical Model I* can be found in [16]. In [16], the system performance was only studied in terms of throughput. In this paper we extend the analysis work to examine more performance metrics. In particular we study system latency, saturation point, packet loss, 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 $0 \le n < \infty 0$ 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* also 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.

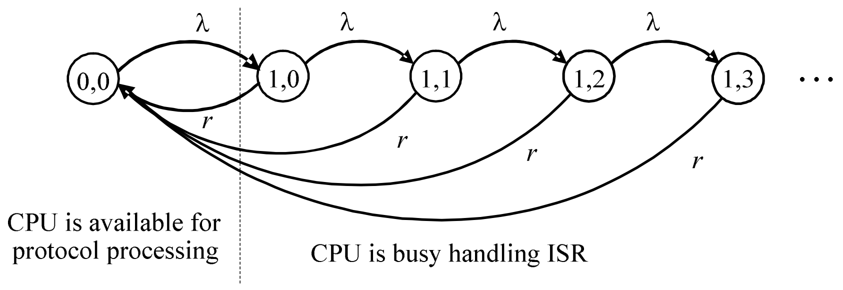


Figure 2. Markov state transition diagram for modeling CPU usage with DMA

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, U_{ISR}, can be expressed as

$$U_{ISR} = \left(\frac{\lambda}{\lambda + r}\right). \tag{1}$$

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

$$\mu' = \mu \times \text{(\% CPU availability for protocol processing)}, \tag{2}$$

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

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

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 (2) that the mean effective service time $1/\mu'$ is exponential. Therefore, the protocol processing can be modeled as *M/M/1/B* queue with a mean service rate of μ'. Hence, the CPU availability for other processing can be expressed as

$$V = \left(\frac{r}{\lambda + r} \right) \cdot p_0 , \tag{3}$$

where p_0 is the probability of not queueing, i.e. finding zero packets, in the *M/M/1/B* queueing system of the kernel's protocol processing.

$$p_0 = \frac{1 - \rho_{IP}}{1 - \rho_{IP}^{B+1}} , \tag{4}$$

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.

CPU Utilization. The CPU utilization, U_{ISR+IP} , which includes ISR handling and protocol processing can be expressed as

$$U_{ISR+IP} = 1 - V \tag{5}$$

The CPU utilization for protocol processing, U_{IIP}, can be determined two ways. Both ways give the same outcome. One way is $U_{IP} = U_{ISR+IP} - U_{ISR}$. Substituting equation (1) and equation (5), U_{IP} can be simplified to $\left(\frac{\lambda}{\lambda + r} \right) \cdot (1 - p_0)$. The other way is to express as the probability of no ISR handling and the probability of queueing, i.e. finding one or more packets, in the *M/M/1/B* queueing system of the kernel's protocol processing. Hence, the CPU utilization for protocol processing, U_{IIP}, can be expressed as

$$U_{IP} = \left(\frac{\lambda}{\lambda + r} \right) \cdot (1 - p_0). \tag{6}$$

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

$$\gamma = \mu'(1 - p_0),$$

where p_0 is expressed by equation (3). Also γ can be expressed as the effective arrival rate λ' which is $\lambda(1 - P_{loss})$. Therefore,

$$\gamma = \mu'(1 - p_0) = \lambda(1 - P_{loss}), \qquad (7)$$

where P_{loss} is the loss probability for a protocol buffer of size B can be expressed as

$$P_{loss} = \frac{(1 - \rho_{IP})\rho_{IP}^{B}}{1 - \rho_{IP}^{B+1}}. \qquad (8)$$

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. 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% with $U_{ISR+IP} = 1$, and thus resulting in $V = 0$. The CPU power is being consumed by ISR handling and protocol processing. The saturation condition can be expressed as

$$V = 0 \quad or \quad \rho_{IP} = 1 \quad or \quad \lambda = \mu'. \qquad (9)$$

Note that when $V = 0$, $\rho_{IP} = 1$. This relation can be derived simply setting V in equation (3) to zero. Therefore, $\left(\frac{r}{\lambda + r}\right) \cdot p_0 = 0$, or $p_0 = 0$. Substituting for p_0 in equation (4), we get $\rho_{IP} = 1$.

Using equation (1), 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 saturation point occurs at

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

Later we will refer to this point as the cliff point or λ_{cliff}. It is to be noted that this equation can also be derived by finding the maximum point of system throughput. This can be done by taking the derivative of the system throughput of equation (6) with respect to λ and setting it to zero, i.e. $\frac{d\gamma}{d\lambda} = 0$, and then solving for λ.

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. In order to find such a delay, we utilize the principles of Jackson theorem for analyzing our queueing model. In particular, we use the approximation method of analyzing queueing models or systems by decomposition discussed in [26]. In this method, the arrival rate must be Poisson and the service times are exponentially distributed, which are the case in our model. Analysis by decomposition is summarized in first isolating the queueing system into subsystems, e.g., single queueing system or process. Next, analyzing each subsystem separately, considering its own surroundings of arrivals and departures. Then, finding the average delay for each individual queueing subsystem. And finally, aggregating all the delays of queueing subsystems to find the average total end-to-end network delay.

Accordingly, the mean system delay is therefore decomposed to be the sum of the mean delay of interrupt handling plus the mean delay of protocol processing. Hence the total mean system delay, $E(r)$, 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 process 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,

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

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 simply the mean delay encountered in the *M/M/1/B* queueing system with $\rho_{IP} = \left(\frac{\lambda}{\mu'}\right)$. According to [27], such delay can be expressed as

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

where $E_{IP}(n) = \frac{\rho_{IP}}{1-\rho_{IP}} - \frac{(B+1)\rho_{IP}^{B+1}}{1-\rho_{IP}^{B+1}}$ and λ' is the mean effective arrival rate. λ' is expressed in equation (7). Therefore, the mean system delay, according to approximation by decomposition method, is

$$E(r) = \frac{1}{r} + \frac{E_{IP}(n)}{\lambda'}. \qquad (11)$$

3.5. Analytical Model II

This model captures the behavior of the interrupt-driven system using only a Markov process. 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.

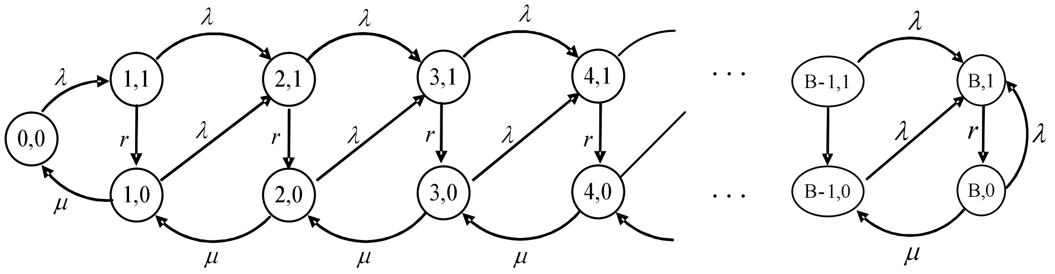


Figure 3. Markov state transition diagram for interrupt-driven system with finite buffer

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:

$$\begin{aligned}
&0 = -\lambda p_{0,0} + \mu p_{1,0}, \\
&0 = -(\lambda + r)p_{1,1} + \lambda p_{0,0}, \\
&0 = -(\lambda + \mu)p_{n,0} + r p_{n,1} + \mu p_{n+1,0} && \text{for } n \ge 1, \\
&0 = -(\lambda + r)p_{n,1} + \lambda p_{n-1,0} + \lambda p_{n-1,1} && \text{for } n \ge 2.
\end{aligned} \qquad (12)$$

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:

$$p_{n+1,0} = \frac{\lambda+\mu}{\mu} p_{n,0} - \frac{r}{\mu} p_{n,1} \qquad n \geq 1,$$

$$p_{n+1,1} = \frac{\lambda}{\lambda+r} p_{n,0} + \frac{\lambda}{\lambda+r} p_{n,1} \qquad n \geq 1.$$

These equations can be written in the vector form as

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

where

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

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

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 [28].

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,

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

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

$$\left. \begin{aligned} p_{n,0} &= \alpha\, p_{0,0}(\alpha+\beta)^{n-1} \\ p_{n,1} &= \beta\, p_{0,0}(\alpha+\beta)^{n-1} \end{aligned} \right\} \quad n \geq 1 \tag{13}$$

The boundary probabilities at state (*B*, *m*) are

$$-(\lambda+\mu)p_{B,0} + r\, p_{B,1} = 0, \tag{14}$$

$$-r\, p_{B,1} + \lambda\, p_{B-1,1} + \lambda\, p_{B-1,0} + \lambda\, p_{B,0} = 0. \tag{15}$$

Substituting equation (15) into (14), we get

$$-(\lambda+\mu)p_{B,0} + \lambda\, p_{B-1,1} + \lambda\, p_{B-1,0} + \lambda\, p_{B,0} = 0,$$

$$\mu\, p_{B,0} = \lambda(p_{B-1,0} + p_{B-1,1}),$$

$$p_{B,0} = \frac{\lambda}{\mu}(p_{B-1,0} + p_{B-1,1}).$$

Using equations (12) to obtain $p_{B-1,0}$ and $p_{B-1,1}$, and then substituting them into the above equation, we get

$$p_{B,0} = \alpha\, p_{0,0}\,(\alpha+\beta)^{B-1}. \tag{16}$$

Now substitute equation (16) into equation (14), we have

$$p_{B,1} = \frac{\lambda}{r}\, p_{0,0}\,(\alpha+1)(\alpha+\beta)^{B-1}. \tag{17}$$

Since the summation of all probabilities is equal to 1, we get

$$p_{0,0} + \sum_{n=1}^{B-1}(p_{n,0}+p_{n,1}) + p_{B,0} + p_{B,1} = 1,$$

$$p_{0,0} + p_{0,0}\sum_{n=1}^{B-1}(\alpha+\beta)^n + p_{0,0}(\alpha + \frac{\lambda}{r}(\alpha+1))(\alpha+\beta)^{B-1} = 1.$$

Therefore,

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

Now, let $\rho = \alpha+\beta$ and $\alpha + \lambda / r \cdot (\alpha+1) = \rho/(1-\beta)$, then

$$p_{0,0} = \frac{1-\rho}{1-\frac{\alpha}{1-\beta}\rho^B}. \tag{18}$$

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

$$\begin{aligned}
U_{ISR} &= \sum_{n=1}^{B} p_{n,1} \\
&= \left(\sum_{n=1}^{B-1} p_{n,1}\right) + p_{B,1} \\
&= \beta\, p_{0,0}\left(\sum_{n=1}^{B-1}\rho^{n-1}\right) + \frac{\beta}{1-\beta}(\alpha+1)\, p_{0,0}\, \rho^{B-1} \\
&= \beta\, p_{0,0}\left(\frac{1-\rho^{B-1}}{1-\rho} + \frac{(\alpha+1)\rho^{B-1}}{1-\beta}\right) \\
&= \beta\, p_{0,0}\left(\frac{(1-\beta) - (1-\beta)\rho^{B-1} + (\alpha+1)(1-\rho)\rho^{B-1}}{(1-\rho)(1-\beta)}\right) \\
&= \beta\, p_{0,0}\left(\frac{1-\beta-\alpha\rho^B}{(1-\rho)(1-\beta)}\right) \\
&= \beta.
\end{aligned}$$

Therefore,

$$U_{ISR} = \left(\frac{\lambda}{\lambda + r} \right).$$

U_{ISR} derived here for *Analytical Model II* is equivalent to equation (1) of *Analytical Model I.*

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

$$\begin{aligned} U_{IP} &= \sum_{n=1}^{B} p_{n,0} \\ &= \alpha\, p_{0,0} \sum_{n=0}^{B} \rho^{n-1} \\ &= \alpha \left(\frac{1-\rho^{B}}{1-\rho} \right) \left(\frac{(1-\beta)(1-\rho)}{1-\beta-\alpha\rho^{B}} \right) \\ &= \frac{1-\rho^{B}}{\frac{1}{\alpha} - \frac{1}{1-\beta}\rho^{B}}. \end{aligned}$$

Therefore,

$$U_{IP} = \frac{1-\rho^{B}}{\frac{1}{\alpha} - \frac{1}{1-\beta}\rho^{B}}.$$

U_{IP} derived here for *Analytical Model II* is not mathematically equivalent to equation (6) of *Analytical Model I*, but are very closely matching, as will be demonstrated numerically and graphically in Section 4, Figure 4b.

The CPU availability for other processing can be expressed as

$$V = p_{0,0}.$$

The CPU utilization for both ISR handling and protocol processing, U_{ISR+IP}, can be expressed as

$$U_{ISR+IP} = 1 - V.$$

Also U_{ISR+IP} is equal to the sum of U_{ISR} and U_{IIP}. As a verification point, it can also be proven that $1 - p_{0,0} = U_{ISR} + U_{IP}$. When simplified, both sides of the equations yield the same term.

$$1 - p_{0,0} = U_{ISR} + U_{IP} = \frac{\rho(1-\beta) - \alpha\rho^{B}}{1-\beta-\alpha\rho^{B}}.$$

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 [27], γ can be expressed as $\mu \sum_{n=1}^{B} p_{n,0}$. Therefore, γ can be derived as follows

$$\begin{aligned}\gamma &= \mu \sum_{n=1}^{B} p_{n,0} \\ &= \mu\,\alpha\, p_{0,0} \sum_{n=1}^{B} \rho^{n-1} \\ &= \mu\,\alpha\, p_{0,0} \times \frac{1-\rho^{B}}{1-\rho}.\end{aligned} \tag{19}$$

γ derived here for *Analytical Model II* is not mathematically equivalent to equation (7) of *Analytical Model I*, but are very closely matching, as will be demonstrated graphically in Section 4, Figure 4a.

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

$$V = 0 \quad or \quad p_{0,0} = 0\,.$$

Substituting in equation (18), we get

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

The saturation point can be solved for λ and can be expressed exactly the same as in *Analytical Model I* given by equation (10). Please note that the term $\lambda/\mu + \lambda/(\lambda + r) = 1$ can be simplified to $\lambda^2 + r\lambda - \mu r = 0$, which is the term used in the derivation of equation (10).

Blocking Probability. The loss probability for a buffer of size B of Figure 3 is the probability of being in either state (B,0) or state (B,1). This can be expressed as

$$\begin{aligned}P_{loss} &= p_{B,0} + p_{B,1} \\ &= \alpha\, p_{0,0}\, \rho^{B-1} + \frac{\lambda}{r}\, p_{0,0}\, (\alpha + 1)\rho^{B-1}\,. \\ &= p_{0,0}\rho^{B-1}(\alpha + \frac{\lambda}{r}(\alpha + 1))\end{aligned}$$

If you let $\alpha + \lambda/r \cdot (\alpha + 1) = \rho/(1-\beta)$, then

$$P_{loss} = p_{0,0}\left(\frac{\rho^{B}}{1-\beta}\right).$$

P_{loss} derived here for *Analytical Model II* is not mathematically equivalent to equation (8) of *Analytical Model I*, but are very closely matching, as will be demonstrated graphically in Section 4, Figure 4d.

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 λ' is the mean effective arrival rate. λ' is expressed in equation (19). However $E(n)$ can be derived as follows

$$\begin{aligned} E(n) &= \sum_{n=1}^{B} n(p_{n,0} + p_{n,1}) \\ &= \sum_{n=1}^{B-1} n(p_{n,0} + p_{n,1}) \quad + \quad B \times (p_{B,0} + p_{B,1}) \\ &= p_{0,0} \sum_{n=1}^{B-1} n\rho^n \quad + \quad \frac{B}{1-\beta} p_{0,0}\,\rho^B \\ &= \left(\frac{\rho}{(1-\rho)^2} - \frac{B(1-\rho)+\rho}{(1-\rho)^2}\rho^B + \frac{B}{1-\beta}\rho^B \right) \times p_{0,0}. \end{aligned}$$

Simplifying the above equation, we get

$$E(n) = \frac{\rho(1-\beta)(1-\rho^B)}{(1-\rho)(1-\beta-\alpha\rho^B)} - \frac{\alpha B}{1-\beta-\alpha\rho^B}.$$

$E(r)$ derived here for *Analytical Model II* is not mathematically equivalent to equation (11) of *Analytical Model I,* but are very closely matching, as will be demonstrated graphically in Section 4, Figure 4c.

3.6. Comparison between the Two Models

Thus far we derived equations for the various performance measures using *Analytical Model I* and *Analytical Model II*. An important question to address is how these derived equations of the two analytical models compare to one another. By examining the equations of both models and as discussed in Section 3.5 and 3.6, we find that the models give exact mathematical equivalence for two performance measures: 1) CPU utilization for ISR handling, and 2) system saturation point. For other performance measures, the derived equations given by the two models are not mathematically equivalent, but very closely matching. This will be demonstrated numerically and graphically in Section 4, as very close matching results were obtained. In fact in the majority of cases, the results were exactly matching.

A key point to notice here is that *Analytical Model II* is an accurate model. It is a pure Markovian process which captures totally the *interaction* between ISR handling and protocol processing. However, *Analytical Model I* is an approximation by decomposition method that introduces somewhat loose coupling of ISR handling and protocol processing, and therefore introduces some error [26]. *Analytical Model I* decomposes the system and focuses on the subsystem or portion of protocol processing. The protocol processing is modeled as a queueing system with an effective service rate. The effect of interrupt disruption is captured by the effective service rate. As will be demonstrated in Section 4, the error introduced by *Analytical Model I* is really negligible when *B* is large, i.e., when utilizing a large buffer size for protocol processing. It was found that when *B* > 50, we obtain very closely matching results. In practice, B is usually much larger than 1000 packets.

On the other hand, 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/B* queueing system [27]. In fact, 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.

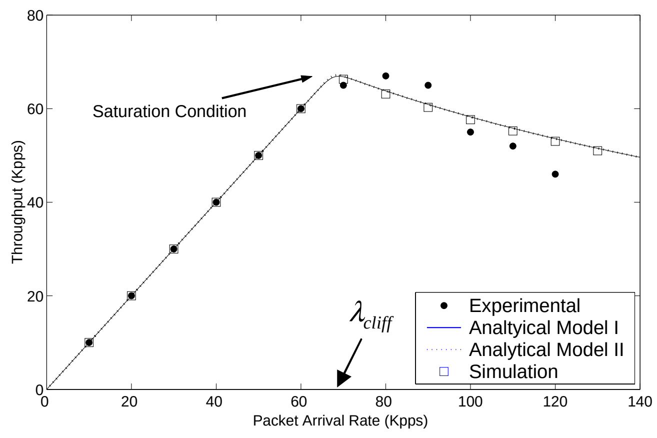
3.7. Simulation

In order to verify and validate our analytical models, a discrete-event simulation model was developed and written in C language. The assumptions of analysis were used. The simulation followed closely and carefully the guidelines given in [29]. We used the PMMLCG as our random number generator [28]. 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 five million 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 [29]. We computed the length of the initial transient period using the MCR (Marginal Confidence Rule) heuristic developed by White [30]. Each replication run lasts for five times of the length of the initial transient period. Analytical and simulation results, as will be demonstrated in Section 4, were very much in line.

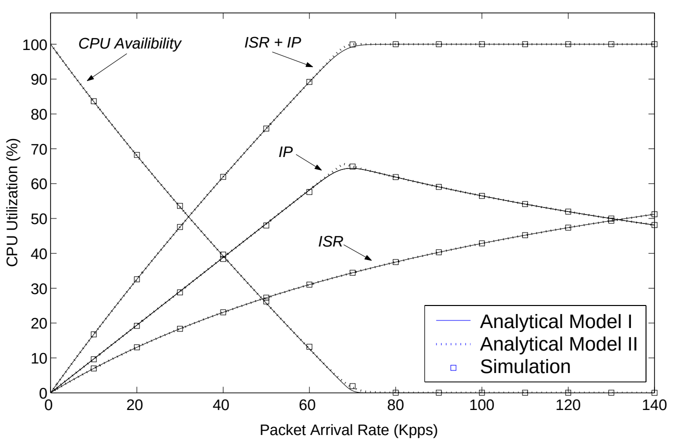
4. Numerical Examples

In this section, we report and compare results of analysis and simulation. Numerical results are given for key performance indicators. These indicators include mean system throughput, CPU utilization, latency, and packet loss probability. For validation, we compare our analysis and simulation results. Also we compare the system throughput to results obtained by lab experiment reported in [5]. For our numerical examples we use the same values for system parameters as those reported in [5]. In [5], 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 [5], 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.

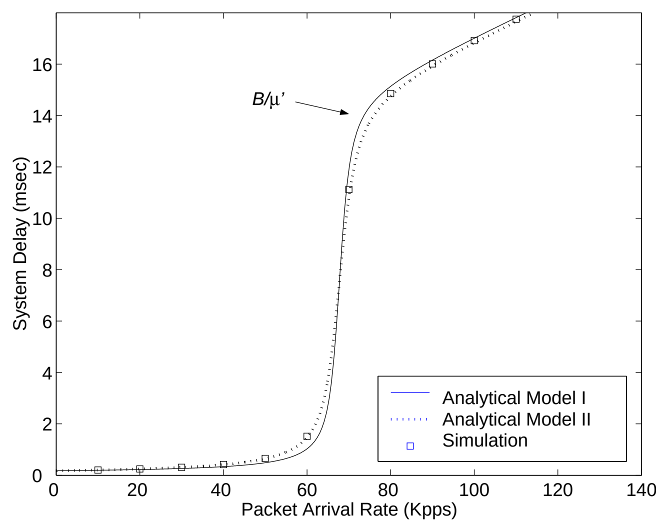
Figure 4a, Figure 4b, Figure 4c, and Figure 4d plot the mean system throughput, CPU utilization, and mean system latency, and packet loss probability, respectively, as a function of packet arrival rate. For Figures 4a, 4b, and 4c, we fix the kernel's protocol processing buffer B to a size of 1000 packets. As for validation, we compare the experimental results for system throughput to those of analysis. Other performance indicators were not measured in [5]. From Figure 4, 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 of system throughput.



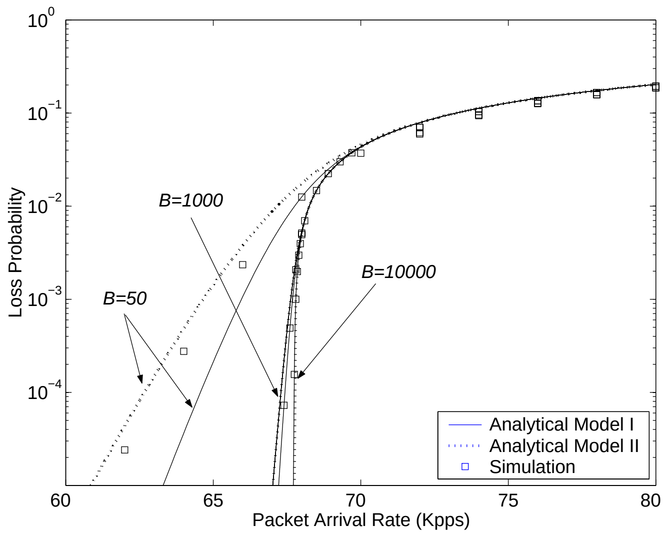
(a)



(b)



(c)



(d)

Figure 4. Key performance indicators in relation to arrival rate

When examining the mean system throughput of Figure 4a and the corresponding CPU utilization and mean system latency of Figure 4b and Figure 4c, it can be noted that the saturation point for the system occurs at 67,750 pps. We referred to the point, where the throughput starts being degraded, as the "cliff" or saturation point. We denoted this point as λ_{cliff}, and was given by equation (10). At this point, the corresponding CPU utilization for both ISR handling and protocol processing is at 100%, with a CPU availability of zero. Therefore, user applications will starve and livelock at this point. Also as expected and depicted in Figure 4b, when the incoming traffic increases beyond the saturation point, the CPU utilization of ISR handling continues increasing whereas the utilization of IP processing starts decreasing. Figure 4c shows the mean system delay sharply increases when reaching the saturation point of an arrival rate of λ_{cliff} = 67,750 pps. Theoretically, the latency should flatten off at B/μ', but rather it slowly shoots to infinity. This is because as the arrival rate increases right after the cliff point, the mean effective service rate μ' decreases. See equation (2).

Figure 4d plots the blocking or loss probability for three values of the kernel's protocol processing buffer *B*: 50, 1000, and 10,000 packets. A key point to notice in the figure is that there is a small difference encountered (close to the saturation condition) between the results obtained by *Analytical Model I* and *Analytical Model II*. However, when *B* is large, the difference is negligible. In practice, this buffer usually holds more than 1000 packets.

Two Important Observations. In general by subjectively eyeballing the curves in all of Figure 4 for *Analytical Model I* and *Analytical Model II*, two important observations can be made. First, it can be observed that both models give very closely matching results. In fact when the buffer size *B* is large, exactly matching results are obtained. Second, *Analytical Model II* is more accurate than *Analytical Model I*. The simulation results are closer to *Analytical Model II.* This is more obvious in Figure 4d when *B=50*. This is due to the fact that *Analytical Model II* is theoretically more accurate than *Analytical Model I*, as discussed in Section 3.6.

5. Design and Implementation Issues

We studied the performance of hosts in terms of mean system throughput, CPU utilization, CPU availability, latency, and packet loss probability when subjected to light and heavy network loads. As noted, degraded and poor performance can be encountered due to heavy network loads. Our work has provided insight into predicting such a system performance. Critical operating points of performance were identified. Based on our analysis, simulation, and experimental studies, the following design recommendations, guidelines and observations 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. The system should be stable even under extremely high load. 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 can also be called the application starvation point. Our analysis provided equations to predict, with adequate degree of accuracy, where this point occurs. We denoted this point as λ_{cliff}, and was given by equation (10).

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 of λ_{cliff}, the host should disable interrupts and enable polling technique. Therefore, interrupt overhead will be eliminated. In polling, the OS periodically polls its host system memory (i.e., protocol processing buffer) to find packets to process. In general, there is a maximum number of packets to process in each poll in order to leave some CPU power for application processing. There are primarily two drawbacks for polling. First, 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. Second, latency of processing the packet is larger, as the packets get queued until they are polled. Therefore, disabling interrupts and enabling polling is only practical at high load. At low load, polling yields excessive latency. And hence, it is only practical to switch to polling mode at overload condition. Switching between interrupts and polling is proposed in [4,10]. However in [4,10], the overload condition was not identified accurately as is the case with our analytical study. In [10], the overload condition was based on the arrival rate and was chosen arbitrarily and has to be tuned manually. Also in [4], the overload condition was based on the host buffer occupancy and two levels of occupancy 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 and protocol processing, λ_{cliff} can be computed. We propose the use of two thresholds of operations: upper (*U*) and lower (*L*), where $U = w\,\lambda_{cliff}$ and $L = z\,\lambda_{cliff}$. w and z are tunable and design parameters, and their value selection depends on how aggressive or releaxed 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 w and z can be 95% and 85%, respectively.

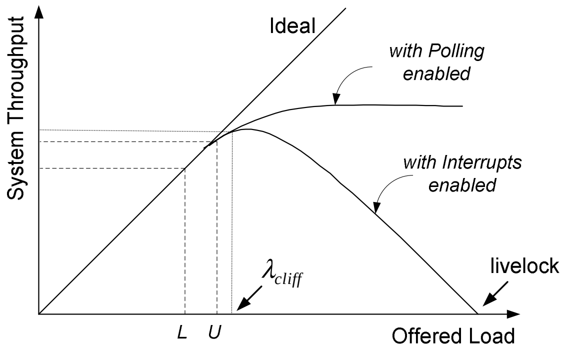


Figure 5. Critical design and operating points

As depicted in Figure 6, 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 [31]. 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 [4]. 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 [4], the upper and lower levels of buffer occupancy were selected arbitrarily. According to [4], determining the proper upper and lower buffer occupancy is an arbitrary and in reality a non-trivial task. These levels vary significantly as they depend on the size of the buffer being used.
- *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 (7) or equation (19). The point of overload condition occurs when $\gamma = \lambda_{cliff}$. Also note that *U* and *L* thresholds for arrival rate is 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.

Maximum Throughput, Latency, and CPU Availability. Our analysis effort provided equations that can be used to easily and quickly predict the host performance and behavior. Given certain known system parameters of protocol 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 and shown in Figure 4a, the maximum system capacity is basically λ_{cliff}, and is given by equation (10). In addition, given a worst-case network load, acceptable performance levels for throughputs, CPU availability, and latency can be reached 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 throughput, CPU availability, and latency. These worst tolerable performance indicators depend on the nature traffic and application. For example real-time applications and traffic such as Voice over IP (VoIP) require a latency of 30ms at the end host [20]. However, non-real time traffic and applications HTTP and FTP tolerate much larger latencies.

Queue Length. One important design issue is selecting the proper size for the kernel's protocol processing buffer. Given input system parameters and a desired packet loss probability, one can determine the proper size of the buffer. For example, given a desired packet blocking or loss probability P_{loss} and other input system parameters such as λ and μ, one can determine the required buffer size *B* for kernel's protocol processing. For *Analytical Model I*, this can be derived from equation (8) as follows

$$P_{loss} \times \left(1-\rho_{IP}^{B+1}\right) = (1-\rho_{IP})\rho_{IP}^{B},$$

$$\rho_{IP}^{B} = \frac{P_{loss}}{1-(1-P_{loss})\rho_{IP}}.$$

Taking the natural logarithm of both sides and solving for B, we get

$$B = \ln\left(\frac{P_{loss}}{1-(1-P_{loss})\rho_{IP}}\right) \Big/ \ln(\rho_{IP}).$$

Similarly, one can also derive *B* from *Analytical Model II*. It is worth noting that the input parameter of packet loss probability depends on the nature of traffic. Real-time traffic such as voice and video tolerates very small (almost no) packet loss, where as data traffic such as FTP and Email tolerates much larger packet loss probability. According to [20], the required VoIP packet loss should be less than 10^{-5}.

6. Conclusion

We developed and validated two analytical models to study and investigate the impact of interrupt overhead caused by Gigabit Ethernet network traffic on OS performance. The two models yielded equations for a number of important system performance metrics. These metrics included system throughput, CPU utilization and availability, latency, and packet loss. The two models yield closed-form solutions and equations that are either mathematically equivalent or very closely matching. In fact when using a large buffer size, exactly matching results can be obtained. As demonstrated in the paper, *Analytical Model II* was shown to be more accurate than *Analytical Model I*. However, *Analytical Model I* is more convenient and can yield more tractable mathematical solution than *Analytical Model II*. *Analytical Model I* is based on queueing system and hence known equations can be directly applied to compute performance metrics. 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. Our analysis effort provided equations that can be used to easily and quickly predict the system and host performance and behavior. The paper also provided design and implementation guidelines and recommendations to improve performance. The two 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, we are currently studying and evaluating the performance of the different proposed schemes for minimizing and eliminating the interrupt overhead caused by heavy network loads.

References

[1] W. Feng, “Is TCP an Adequate Protocol for High-Performance Computing Needs?” *Proceedings of SC2000*, Dallas, Texas, USA, November 2000.

[2] A. Foong, T. Huff, H. Hum, J. Patwardhan, and G. Regnier, “TCP Performance Re-Visited,” IEEE Symposium on Performance of Systems and Software, March 2003, pp. 70-79

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

[4] 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.

[5] 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.

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

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

[8] 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.

[9] 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.

[10] 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.

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

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

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

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

[15] 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.

[16] 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

[17] 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.

[18] 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.

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

[20] M. Karam and F. Tobagi, “Analysis of Delay and Delay Jitter of Voice Traffic in the Internet,” *Computer Networks Magazine*, vol. 40, no. 6, December 2002, pp. 711-726.

[21] 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.

[22] V. Paxson and S. Floyd, “Wide-Area Traffic: The Failure of Poisson Modeling,” IEEE/ACM Transactions on Networking, vol. 3, no. 3, June 1995, pp. 226-244

[23] 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.

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

[25] 3Com Corporation, "Gigabit Server Network Interface Cards 7100xx Family,” http://www.costcentral.com/pdf/DS/3COMBC/DS3COMBC109285.PDF

[26] K. M. Chandy and C. H. Sauer, “Approximate methods for analyzing queueing network models of computing systems,” *Journal of ACM Computing Surveys*, vol. 10, no. 3, September 1978, pp. 281-317.

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

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

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

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

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