# TWO ANALYTICAL MODELS FOR EVALUATING PERFORMANCE OF GIGABIT ETHERNET HOSTS

**Khaled Salah***

*Department of Information and Computer Science*
*King Fahd University of Petroleum and Minerals*
*PO Box 5066, Dhahran 3126, Saudi Arabia*

**الخلاصة:**

تم العمل على تطوير نموذجين تحليليين لدراسة تأثير طلبات المقاطعة على أداء نظم تشغيل الأجهزة المتصلة بشبكة جيجابت إثرنت عند استقبالها لسيل البيانات من الشبكة، حيث تؤثر طلبات المقاطعة الناتجة عن استقبال البيانات على الأداء بشكل ملحوظ خاصة عندما تزداد سرعة ورودها، ويظهر ذلك خصوصا على شكل زيادة في وقت المعالجة وتقليل لانتاجية النظام، كذلك يؤدي استنفاد طاقة المعالج في الاستجابة لطلبات المقاطعة إلى التأثير على عمل التطبيقات الأخرى خاصة التفاعلية منها. يقدم هذا البحث نموذجين تحليليين لأداء الأجهزة في تلك الظروف ويقارن بينهما، أول هذين النموذجين يستعمل نماذج ماركوف ونظرية الطوابير بينما يعتمد الثاني كلية – وهو النموذج الأدق والأكثر تعقيدا - على نماذج ماركوف، وينتج النموذجان صيغا رياضية متقاربة في معظم الحالات لعدد من المقاييس المهمة لأداء النظام ومنها: انتاجية النظام، واستقراره، والوقت الذي تستغرقه المعالجة، ومعدل انشغال المعالج بطلبات المقاطعة، والوقت المتاح للتطبيقات الأخرى. تمكن هذه النماذج من فهم وتوقع آثار الاختيارات التي تتخذ عند تصميم الأنظمة والشبكات على أداء الأنظمة المدارة بالمقاطعة تحت أحمال مختلفة للشبكة ، كما تفتح المجال لتحسين أداء الأنظمة وذلك باتباع النصائح والمقترحات الواردة في هذا البحث والتي تشمل كلا مرحلتي التصميم والتنفيذ، كذلك يورد البحث نتائج المحاكاة والتجارب التي تدل على صحة ودقة النماذج المقدمة.

[*] e-mail: salah@kfupm.edu.sa

**Paper Received 8 December 2004; Revised 25 June 2005; Accepted 10 October 2005**

## ABSTRACT

Two analytical models are developed to study the impact of interrupt overhead on operating system performance of network 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 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. For the most part both models give mathematically-equivalent closed-form solutions for a number of important system performance metrics. These metrics include 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. 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.

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

# TWO ANALYTICAL MODELS FOR EVALUATING PERFORMANCE OF GIGABIT ETHERNET HOSTS

## 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 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 exceeds 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 $\mu$s 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 $\mu$s. Also, a typical latency for handling interrupt due to a packet arrival in Linux is in the order of 50 $\mu$s. 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 (Interrupt Service Routine) 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 sharp contrast to early work presented in [16,17], this paper is different in significant ways. Reference [16] is a short paper and presents only a preliminary simple throughput analysis for interrupt-driven kernels when utilizing PIO (Programmed Input/Output) and DMA. (Direct Memory Access) [17] focuses on DMA systems and studies latency and system power. In this paper, we present *two* equivalent analytical model to study a number of important performance metrics. One model is simple and the other is more complex. *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. *Analytical Model II* is more complex as it is based on pure Markovian processes. *Analytical Model II* is completely novel. In this paper, we substantially expand and study the performance in terms 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. One major contribution of the paper is proving that these two models are mathematically equivalent, *i.e.*, both yield the same analytical results. Moreover, the paper gives guidelines and recommendations to address design and implementation issues.

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 (Network Interface Card) and system parameters. These parameters may include the proper service times for ISR handling and protocol processing, buffer sizes, CPU bandwidth allocation for protocol process and application, etc.

The rest of the paper is organized as follows. Section 2 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 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 a rate is an acceptable rate and is relatively flat after that.

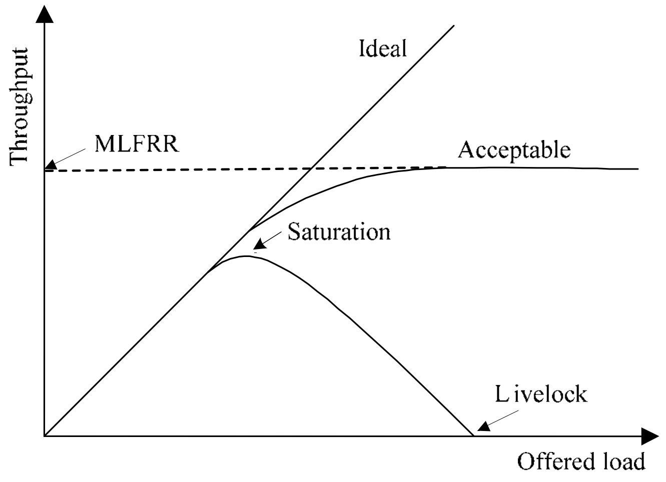


*Figure 1. Receive livelock phenomenon.*

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 $\lambda$ be the mean incoming packet arrival rate and $\mu$ 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 [7–9,14,18–20]. 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 [21], 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, systems having variable-size packets, *e.g.* Jumbo frames, and other traffic distributions, *e.g.* bursty traffic [22,23], 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 [24]. 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 [25, 26].

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.

One might think that such an interrupt-driven system can be simply modeled as a priority queueing system with preemption in which there are two arrivals of different priorities. The first arrival is the arrival of ISRs, and has the higher priority. The second arrival is the arrival of incoming packets, and has the lower priority. As noted the ISR execution preempts protocol processing. However this is an invalid model because ISR handling is not counted for every packet arrival. ISR handling is ignored if the system is servicing another interrupt of the same level. In other words, if the system is currently executing another ISR, the new ISR which is of the same priority interrupt level will be masked off and there will be no service for it. We use instead two analytical models: one is based on an *M/G/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 an *M/G/1* queueing system with this mean effective rate. More details on *Analytical Model I* can be found in [16,17]. 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 $0 \le n < \infty$ represent 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.

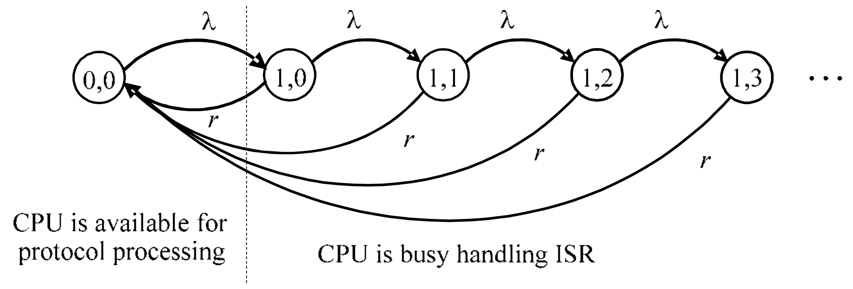


*Figure 2. Markov state transition diagram for modeling CPU usage with DMA.*

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 is $\dfrac{\lambda}{\lambda + r}$. The mean effective service rate $\mu'$ 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 $1 - \dfrac{\lambda}{\lambda + r}$. Please note that by examining Equation (1) the disrupted or effective service time with a mean $1/\mu'$ is exponentially distributed since the service time with a mean of $1/\mu$ is exponential, and thus the *M/G/1* queueing model becomes an *M/M/1* model.

**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. As noted from Equation (1), the effective service time is exponential. Therefore, the protocol processing can be modeled as an *M/M/1/B* queue with a mean service rate of $\mu'$. 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 $\rho_{IP} = \left(\dfrac{\lambda}{\mu'}\right)$. Note that $\rho_{IP}$ 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 $\rho$ 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 $\frac{\lambda}{\mu}$. The CPU utilization for protocol processing can also be thought of as the probability of no ISR handling and having one or more packets being processed by the protocol stack, *i.e.* $\frac{r}{\lambda + r} \cdot \left( \frac{\lambda}{\mu'} \right)$, 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 $\lambda$ can be done in 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 $\lambda$ 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

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

Later we will refer to this point as the cliff point or $\lambda_{cliff}$.

**Mean System Throughput.** The mean system throughput $\gamma$ 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. 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 [27]. 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 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

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

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 is simply the mean delay encountered in the *M/M/1* queueing system with $\rho_{IP} = \left(\frac{\lambda}{\mu'}\right)$. According to [28], 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 the approximation by decomposition method, is

$$E(r) = \frac{1}{r} + \frac{1}{\mu' - \lambda}\,. \tag{7}$$

## 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 $S = \{\,(n, m),\, 0 \le n < \infty,\, m \in \{0,1\}\,\}$, 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.

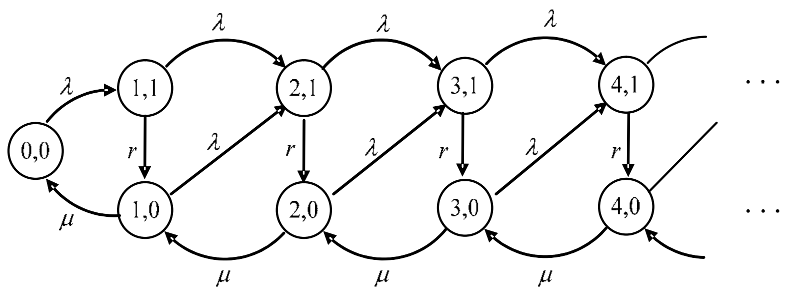


*Figure 3. Markov state transition diagram for interrupt-driven system with DMA*

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:

$$\begin{aligned} &0 = -\lambda p_{0,0} + \mu p_{1,0}\,, \\ &0 = -(\lambda + r)p_{1,1} + \lambda p_{0,0}\,, \\ &0 = -(\lambda+\mu)p_{n,0} + rp_{n,1} + \mu p_{n+1,0} && \text{for } n \ge 1\,, \\ &0 = -(\lambda + r)p_{n,1} + \lambda p_{n-1,0} + \lambda p_{n-1,1} && \text{for } n \ge 2\,. \end{aligned} \tag{8}$$

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:

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

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

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 the Putzer algorithm to obtain the solution [29].

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 the 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

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

The solution of the difference equation is given by

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

The solution can be nicely simplified to

$$\left.\begin{aligned} p_{n,0} &= \alpha\, p_{0,0}(\alpha+\beta)^{n-1} \\ p_{n,1} &= \beta\, p_{0,0}(\alpha+\beta)^{n-1} \end{aligned}\right\} \quad n \ge 1 \tag{9}$$

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

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

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

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

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 $\rho$ 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 (10) 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 $\lambda$ and can be expressed exactly as in *Analytical Model I* given by Equation (4).

**Mean System Throughput.** The mean system throughput, $\gamma$, for the pure Markovian model is the rate at which packets are successfully being processed by the kernel's protocol stack. According to [28], $\gamma$ can be expressed as $\mu\sum_{n=1}^{\infty} p_{n,0}$.

Therefore, $\gamma$ can be derived as follows

$$\gamma = \mu \sum_{n=1}^{\infty} p_{n,0} = \mu \sum_{n=1}^{\infty} \alpha (1-\rho)\, \rho^{n-1} = \mu\alpha(1-\rho) \sum_{n=1}^{\infty} \rho^{n-1} = \mu\alpha(1-\rho) \times \frac{1}{1-\rho} = \lambda\,. \tag{11}$$

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, this equation is not mathematically equivalent to Equation (7) of *Analytical Model I.* As it will be demonstrated in Section 4 when giving numerical examples, both equations yield very close matching results. In fact at light load, the equations yield exactly matching results.

### 3.6. Comparison Between the Two Models

From Section 3.4 and Section 3.5, 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 4 and Figure 4*c*, 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. 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 [27]. *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.

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 [28]. 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 [30]. We used the PMMLCG as our random number generator [30]. 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 [30]. We computed the length of the initial transient period using the MCR (Marginal Confidence Rule) heuristic developed by White [31]. 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 the mean system throughput, latency, and CPU utilization. For validation, we compare our analysis and simulation results of system throughput to lab experimental results 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. 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 μs and the mean protocol processing time ($1/\mu$) was 9.7 μs.

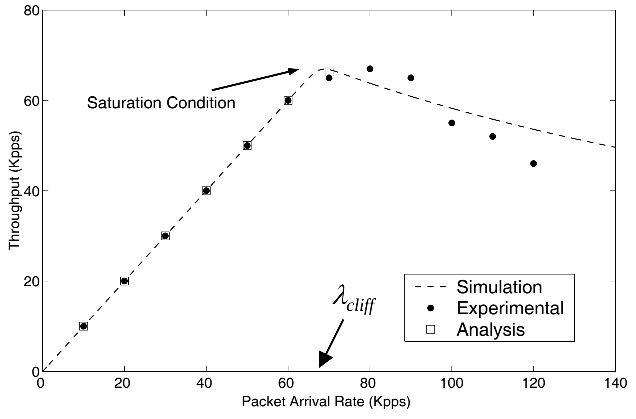
Figure 4(*a*), Figure 4(*b*), and Figure 4(*c*) plot the mean system throughput, CPU utilization, and mean system latency, respectively, as a function of packet arrival rate. Analysis results are only plotted when the system is stable, *i.e.* up to the saturation point. As for validation, we compare 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 [5]. 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 4(*a*) 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.

When examining the mean system throughput of Figure 4(*a*) and the corresponding CPU utilization and mean system latency of Figure 4(*b*) and Figure 4(*c*), 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 $\lambda_{cliff}$, and it was given by Equation (5). 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. In addition, the mean system delay will shoot rapidly to infinity, when reaching the saturation point of an arrival rate of $\lambda_{cliff}$ = 67.750 pps, causing excessive latency and timeouts. It can also be observed from Figure 4(*b*) that if the incoming traffic increases beyond the staturation point, as captured by the simulation, the CPU utilization of ISR handling continues increasing whereas the utilization of IP processing starts 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.

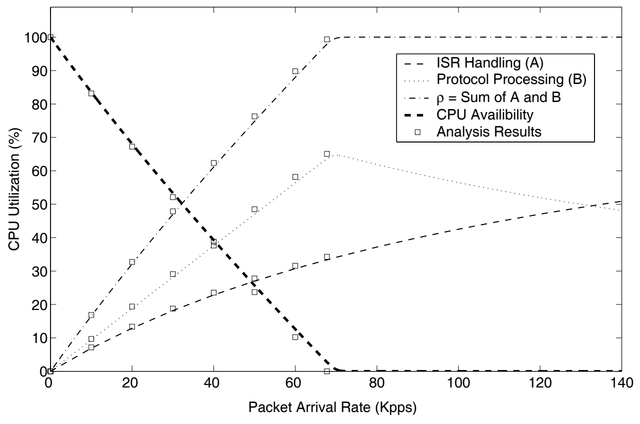
Figure 4(*c*) compares the latency results of *Analytical Model I* with those of *Analytical Model II*. It is depicted in the figure that both models give very close matching results. The results are exact in the light load and are very closely matching around the saturation point of the system. The results of both analytical models for other performance metrics are not plotted because they are mathematically equivalent, as discussed in Section 3.6.

## 5. DESIGN AND IMPLEMENTATION ISSUES

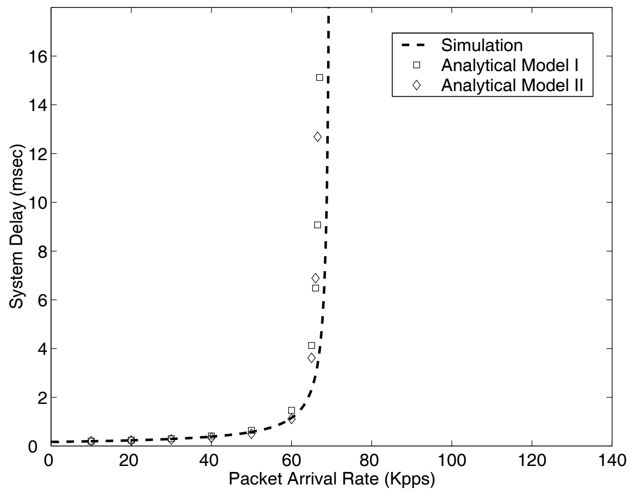
We studied the performance of hosts in terms of mean system throughput, CPU utilization, CPU availability, latency, and packet loss probability when subjected to light and heavy network loads. As noted, degraded and poor performance can be encountered due to heavy network loads. Our work has provided 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.



(a)



(b)



(c)

*Figure 4. Mean system throughput, CPU utilization, and mean system latency*

**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 $\lambda_{cliff}$, and it was given by Equation (5).

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 $\lambda_{cliff}$, the host should disable interrupts and enable a 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 the 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 $\lambda_{cliff}$. Given the system parameters of interrupt overhead and protocol processing, $\lambda_{cliff}$ 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 relaxed the need of switching between interrupts and polling. The value selection also depends on the CPU availability percentage required to be reserved for application processing. Good design values for $w$ and $z$ can be 95% and 85%, respectively.

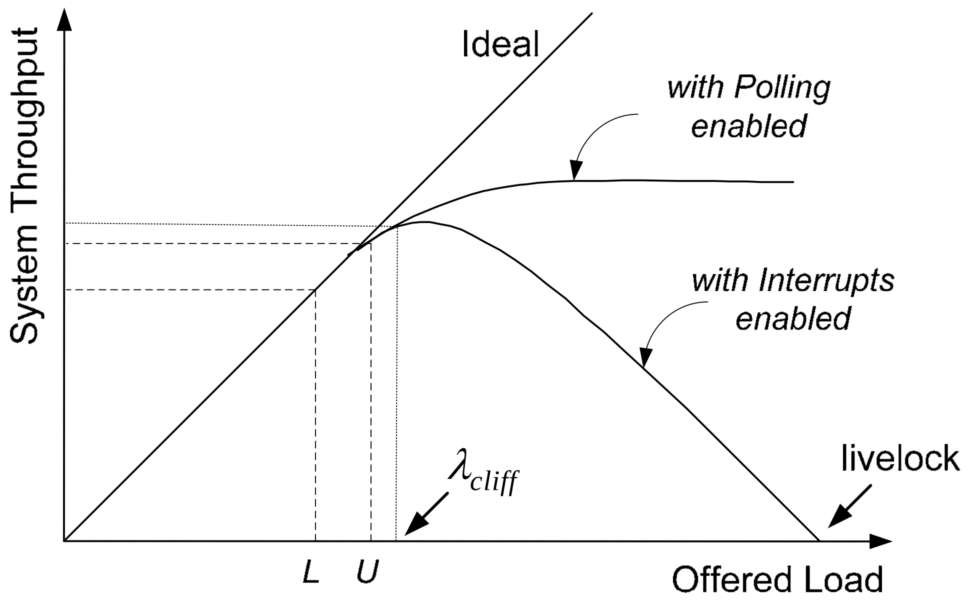


*Figure 5. Critical design and operating points*

As depicted in Figure 5, 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 $\lambda$ exceeds the upper threshold $U$, the host's OS must switch to polling mode. When the arrival rate $\lambda$ 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 $\lambda$ by recording and measuring the inter-arrival times of incoming packets using exponential averaging method as reported in [32]. 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 become 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 $\gamma$ by Equation (9) or Equation (11). The point of overload condition occurs when $\gamma = \lambda_{cliff}$. Also note that $U$ and $L$ thresholds for arrival rate are the same $U$ and $L$ thresholds for system throughput. Hence, when $\gamma > U$, the OS must switch to polling mode. When $\gamma < L$, the OS must switch to interrupts mode. This method is as accurate as that of the NIC-side solution, however this method requires more overhead on the part of the OS. The OS can use a 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 4($a$), the maximum system capacity is basically $\lambda_{cliff}$, and is given by Equation (5). 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 of the and application. For example real-time applications and traffic such as Voice over IP (VoIP) require a latency of 30ms at the end host [21]. However, non-real time traffic and applications HTTP and FTP tolerate much larger latencies.

## 6. CONCLUSION

We developed two analytical models to study the impact of interrupt overhead caused by Gigabit Ethernet network traffic on OS performance. For the most part, the two models give mathematically equivalent closed-form solutions for performance. As demonstrated in the paper, *Analytical Model I* is more convenient and can yield more tractable mathematical solutions than *Analytical Model II*. *Analytical Model I* is based on the *M/M/1* queueing system and hence known equations can be directly applied to compute different performance metrics. *Model I* can be applicable to model and analyze other similarly behaving systems that get distracted by higher priority events or activities. Determining the distraction percentage, one can then use straightforward queueing system equations. In fact, *Analytical Model I* is currently being utilized by the author to evaluate the performance of interrupt-handling schemes such as interrupt coalescing and polling. Using these two 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. 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.

## ACKNOWLEDGEMENTS

The author wishes to thank Mr. Khalid ElBadawi 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] 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*, **11**(**2**) (1993), pp. 203-219.

[4] J. Mogul, and K. Ramakrishnan, “Eliminating Receive Livelock In An Interrupt-Driven Kernel,” *ACM Trans. Computer Systems,* **15**(**3**) (1997), pp. 217-252.

[5] R. Morris, E. Kohler, J. Jannotti, and M. Kaashoek, “The Click Modular Router,” *ACM Transactions on Computer Systems*, **8**(**3**) (2000), pp. 263-297.

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

[7] P. Druschel, “Operating System Support for High-Speed Communication,” *Communications of the ACM*, **39**(**9**) (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*, **35** (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*, **11**(**2**) (1993), pp. 240-253

[14] C. Traw. and J. Smith, "Giving Applications Access to Gb/s Networking," *IEEE Network*, **7**(**4**) (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. ElBadawi, " Evaluating System Performance in Gigabit Networks ", *The 28th IEEE Local Computer Networks (LCN), Bonn/Königswinter, Germany*, October 20–24, 2003, pp. 498-505

[17] K. Salah and K. Badawi, " Performance Evaluation of Interrupt-Driven Kernels in Gigabit Networks", The IEEE Global *Telecommunications Conference, 2003, GLOBECOM'03*, December 1-5, 2003, pp. 3953-3957

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

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

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

[21] M. Karam and F. Tobagi, "Analysis of Delay and Delay Jitter of Voice Traffic in the Internet," *Computer Networks Magazine*, **40** (**6**) (2002), pp. 711-726.

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

[23] V. Paxson and S. Floyd, "Wide-Area Traffic: The Failure of Poisson Modeling," *IEEE/ACM Transactions on Networking*, **3**(**3**) (1995), pp. 226-244

[24] P. Steenkiste, "A Systematic Approach to Host Interface Design for High-Speed Networks," *IEEE Computer Magazine*, **24**(**3**) (1994), pp. 44-52.

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

[26] 3Com Corporation, "Gigabit Server Network Interface Cards 7100xx Family," http://www.costcentral.com/pdf/DS/3COMBC/DS3COMBC109285.PDF

[27] K. M. Chandy and C. H. Sauer, "Approximate Methods for Analyzing Queueing Network Models of Computing Systems," *Journal of ACM Computing Surveys*, **10**(**3**) (1978), pp. 281-317.

[28] L. Kleinrock, *Queueing Systems: Theory*, vol 1. New York : Wiley, 1975.

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

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

[31] J. White, "An Effective Truncation Heuristic for Bias Reduction in Simulation Output," *Simulation Journal*, **69**(**6**) (1997), pp. 323-334

[32] A. Silberschatz, P. Galvin, and G. Gagne, *Operating System Concepts*, New York: John Wiley & Sons, Inc, 4th Edition, 2003.