Evaluating System Performance in Gigabit Networks

K. Salah** K. El-Badawi
Department of Information and Computer Science
King Fahd University of Petroleum and Minerals
Dhahran 31261, Saudi Arabia
Email: {salah,badawi}@kfupm.edu.sa

Abstract *- With the current wide deployment of Gigabit Ethernet technology in the backbone and workgroup switches, the network performance bottleneck has shifted for the first time in nearly a decade from the network to the end hosts and servers. This dramatic bandwidth increase calls for optimizations and good design considerations in many key components of the hosts and servers. These key components include network adaptor, operating system, protocol stack, memory, and processing power. More importantly the high bandwidth increase can negatively impact the OS performance due to the interrupt overhead caused by the incoming gigabit traffic. This paper presents models and analytical techniques for studying such a negative impact. We first present an analytical model for the ideal system when interrupt overhead is ignored. We then present two models which describe the impact of high interrupt rate on system throughput. One model is for network adaptors not equipped with DMA engines, and the other model is for network adaptors equipped with DMA engines. In addition we study the system performance when using different system delivery options of packet data to user applications. Results from both simulations and reported experimental findings show that our analytical models are valid and give a good approximation.*

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

1 Introduction

Interrupt overhead of high-speed network devices can have a significant negative impact on system performance. Traditional operating systems were designed to handle network devices that interrupt on average rate of around 1000 packets per second, as is the case for shared 10Mbps Ethernet [1]. The cost of handling interrupts in these traditional systems was low enough that any normal system would spend only a fraction of its CPU time handling interrupts.

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

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

The receive livelock condition was shown by experiments and measurements in real systems [2,3]. A number of solutions have been proposed to minimize the interrupt overhead and resolve receive livelock condition. Such solutions include interrupt coalescing, OS-bypass protocol, zero-copy, jumbo frames, polling, pushing some or all

*** Corresponding Author: Prof. K. Salah. Department of Information and Computer Science. King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia. Email: salah@kfupm.edu.sa* protocol processing to hardware, etc. Some of these solutions are listed in [1,3,4,5,6,7,8]. However none of these solutions or others, to the best of our knowledge, modeled and studied analytically the system performance under heavy network loads.

In this paper we present analytical models to study the receive livelock phenomenon. These models can be utilized to understand and predict the performance and behavior of interrupt-driven systems and can be served as a reference model for comparing the performance of proposed solutions to resolve the receive livelock condition. More importantly, the paper presents an analytical study of OS performance in terms of system throughput due to high rate of interrupts found in high speed networks.

The rest of the paper is organized as follows. Section 2 describes the receive livelock phenomenon reported in literature. Section 3 presents analysis for three models: an ideal system that ignores the impact of interrupts on system performance, a second model that describes the system behavior under low and high network traffic when not using DMA, and a third model that describes system behavior when using DMA. Numerical examples are given in Section 4. A note on the accuracy of the analysis is given in Section 5. Finally, Section 6 has the conclusion and identifies future work.

2 Receive Livelock

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

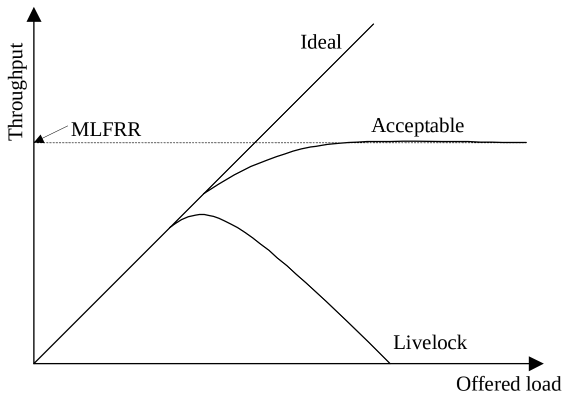

Figure 1. Receive livelock phenomenon

Under network input overload, a host can be swamped with receiving 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 and freeing up CPU processing power for other user tasks or threads before the arrival of the next packet. However, if another packet arrives before completing of handling the first one (e.g., in the case of high packet arrival rate), starvation will occur for user tasks and threads resulting in unpleasant performance of dropping packets due to queue overflows, excessive network latency, and bad system throughput.

3 Analysis

In this section we present an analytical study to examine the impact of interrupts on system performance. At first we define system parameters. Let λ be the mean incoming packet arrival rate, and μ be the mean protocol processing rate carried out by the kernel. Therefore $1/\mu$ is the time it takes the system to process the incoming packet and deliver it to the application program. This time includes primarily the network protocol stack processing carried out by the kernel, excluding any interrupt handling. However, the interrupt handling time will be denoted as T_{ISR}, which is basically the interrupt service routine time for handling incoming packet. We will also define $\rho = \lambda/\mu$. ρ is as a measure of the traffic intensity or system load. We study the system performance in terms of system throughput. System throughput (γ) is the rate at which packets are delivered by the kernel to the application program.

3.1 Ideal System

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

Note. It is worth mentioning that in our analysis we assume a Poisson arrival for network traffic. While it is true that network traffic is bursty in nature, it becomes very difficult to study, model, and solve analytically a system based on such network traffic. As we will demonstrate in Section 4 and 5, it turns out that our model with those assumptions including that of a Poisson arrival is a good approximation to an experimental model with real network traffic. The impact of having bursty traffic instead of Poisson is currently being studied by the authors using simulation and will be reported in the near future.

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

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

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

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

3.2 Network Adaptors With No DMA Engines

Traditional network adaptors or Network Interface Cards (NICs) as those of 10Mbps Ethernet and 16Mbps Token Ring are not equipped with DMA engines. The copying of an arrived packet from NIC buffer to host kernel memory is performed by the CPU as part of ISR handling for each incoming packet. The ISR then sets a software interrupt to trigger packet protocol processing. It is very possible that one or more incoming packets arrive during the execution of the ISR. For this, the ISR handling must not exit unless all incoming packets are copied from the NIC to the kernel memory. When the network packet processing is triggered, the kernel processes the incoming packet by executing the network protocol stack and delivers the packet data to user application [3].

There are two possible system delivery options of packet data to user applications. The first option is to have an extra copy of packet data 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. The kernel is written such that the packet data is delivered to the application using pointer manipulations [5,6]. We will study the impact of both these copy options on system performance.

After the notification of the arrival of a new packet, the kernel will process the packet by first examining the type of frame being received and then invoking immediately the proper handling stack function or protocol, e.g. ARP, IP, TCP, etc. The packet will remain in the kernel or system memory until it is discarded or delivered to the user application. The network protocol processing for packets carried out by the kernel will continue as long as there are packets available in the system memory buffer. However, this protocol processing of packets can be interrupted by ISR executions as a result of new packet arrivals. This is so because packet processing by the kernel runs at a lower priority than the ISR.

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

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

As more packets arrive during ISR handling, i.e. the arrival rate λ becomes high, most of the CPU power will be consumed by ISR handling. And therefore, the rate of protocol processing for incoming packets carried out by the kernel will be degraded to zero.

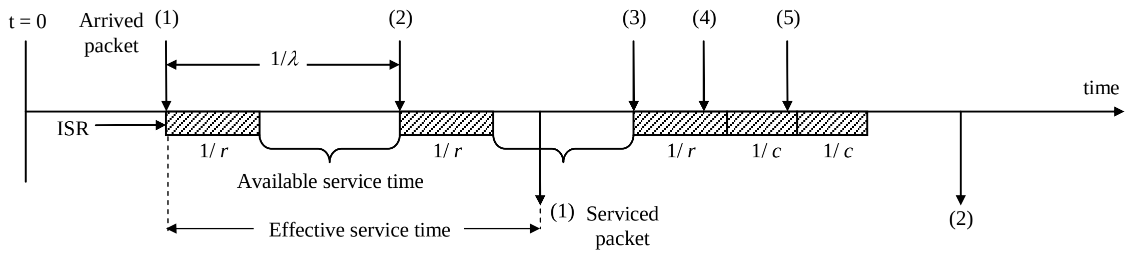

Figure 2. Effective service time with no DMA

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

μ' = Rate at which packets get processed by the kernel's network protocol with no interrupt disruption.

Therefore,

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

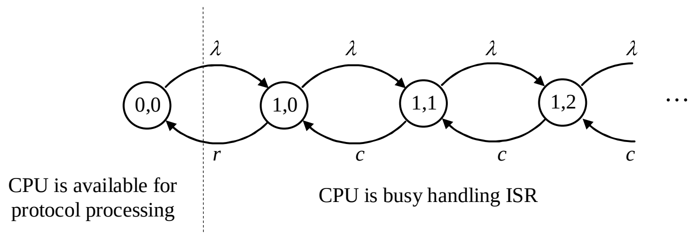

Figure 3. Markov state transition diagram for modeling CPU usage with no DMA

In order to determine the CPU availability percentage for protocol processing and interrupt handling, we use a Markov process to model the CPU usage, as illustrated in Figure 3. The process has state (0,0) and states (1,*n*). State (0,0) represents the state where the CPU is available for protocol processing. States (1,*n*) with $0 < n < \infty$ represents the state where the CPU is busy handling interrupts. *n* denotes the number of packet arrivals or interrupts that are batched or masked off during T_{ISR}. In other words, state (1,0) means there are no interrupts being masked off and the CPU is busy handling an ISR with one packet arrival. State (1,1) means that one interrupt has been masked off and the CPU is busy handling an ISR with two packet arrivals: one packet will be serviced with a mean rate of *r* and the other with a mean rate of *c*.

The steady-state solution of the Markov chain, shown in Figure 3, can be expressed as

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

Using the boundary condition that $p_{0,0} + \sum_{n=0}^{\infty} p_{1,n} = 1$, we get

$$p_{0,0} + \sum_{n=0}^{\infty} \frac{\lambda^{n+1}}{r\,c^n}\, p_{0,0} = 1.$$

Hence,

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

The geometric series $\sum_{n=0}^{\infty} (\lambda / c)^n$ converges only for $\lambda < c$. For $\lambda > c$, the geometric series does not converge, i.e., goes to infinity, and $p_{0,0}$ becomes zero. Thus the CPU availability percentage for protocol processing ($p_{0,0}$) can be expressed as

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

And hence, the mean effective service rate is expressed as

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

It is to be noted from equation (4) that the mean effective service time $1/\mu'$ is exponential. Therefore, the system can be modeled as M/M/1/B queue as is the case for the ideal system. Therefore, the system throughput can be expressed by equation (1). However, the mean service rate μ for equation (1) will have to be replaced by the mean effective service rate μ' of equation (7).

3.3 Network Adaptors with DMA Engines

In order to minimize CPU cycles consumed in copying packets from the NIC to kernel memory, major network vendors equip high-speed NICs with DMA engines. These vendors include 3Com, HP, Alteon owned now by Nortel, Sundace, and NetGear. NICs are equipped with a receive Rx DMA engine and a transmit Tx DMA engine. A Rx DMA engine handles transparently the movement of packets from the NIC internal buffer to the host system memory. A Tx DMA engine handles transparently the movement of packets from the host memory to the NIC internal buffer.

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

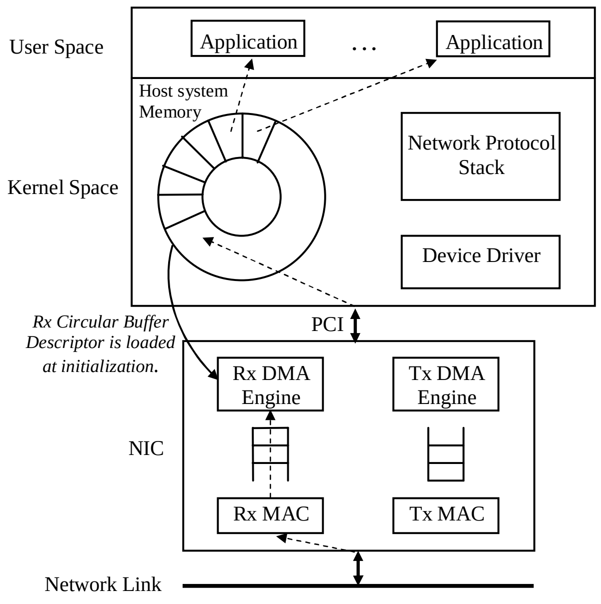

Figure 4. System Architecture Model and Flow of Arrived Packets

It is important to note that the device driver for the network adaptors is typically configured such that an interrupt is generated after the incoming packet has completely moved into the host system memory. In order to minimize the time for ISR execution, ISR handling mainly sets a software interrupt to trigger the protocol processing for the incoming packet. Please note in this situation if two or more packets arrive during an ISR handling, the ISR time for servicing all of these packets will be the ISR time for servicing a single packet, with no extra time introduced. Figure 5 illustrates the ISR handling for incoming packets 3 and 4 is only the ISR handling for packet 3.

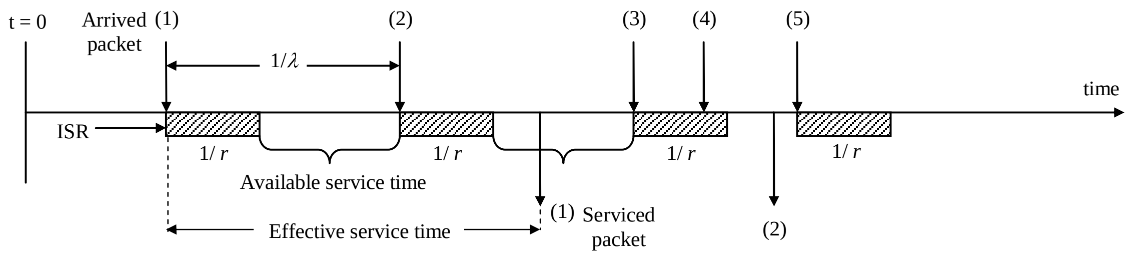

Figure 5. Effective service time with DMA

This system can also be modeled as an M/G/1/B queue with a Poisson packet arrival rate of λ and a mean effective service time of $1/\mu'$ that has a general distribution. In order to determine the mean effective service time $1/\mu'$, we need to determine the CPU availability percentage for protocol processing and interrupt handling. We use a Markov process to model the CPU usage, as illustrated in Figure 6. The process has state (0,0) and states (1,*n*). State (0,0) represents the state where the CPU is available for protocol processing. States (1,*n*) with $0 < n < \infty$ represents the state where the CPU is busy handling interrupts. *n* denotes the number of packet arrivals or interrupts that are batched or masked off during T_{ISR}. Note that state (1,0) means there are no interrupts being masked off and the CPU is busy handling an ISR with one packet arrival. State (1,1) means that one interrupt has been masked off and the CPU is busy handling an ISR with two packet arrivals. Both of these packets will be serviced together with a mean rate *r* of servicing only one packet.

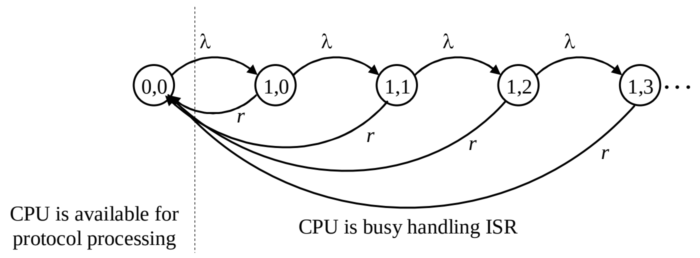

Figure 6. Markov state transition diagram for modeling CPU usage with DMA

The steady-state difference equations can be derived from $\mathbf{0} = \boldsymbol{pQ}$, where $\boldsymbol{p} = \{p_{0,0}, p_{1,0}, p_{1,1}, p_{1,2}, \cdots\}$ and $\boldsymbol{Q}$ is the rate-transition matrix and is defined as follows:

$$\boldsymbol{Q} = \begin{bmatrix} -\lambda & \lambda & 0 & 0 & 0 & \cdots \\ r & -(\lambda+r) & \lambda & 0 & 0 & \cdots \\ r & 0 & -(\lambda+r) & \lambda & 0 & \cdots \\ r & 0 & 0 & -(\lambda+r) & \lambda & \cdots \\ r & 0 & 0 & 0 & -(\lambda+r) & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \end{bmatrix}$$

This will yield

$$-\lambda p_{0,0} + r\,(p_{1,0} + p_{1,1} + p_{1,2} + \cdots) = 0.$$

Since we know that $p_{0,0} + \sum_{n=0}^{\infty} p_{1,n} = 1$, then

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

Solving for $p_{0,0}$, we thus have

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

and

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

Therefore, the CPU availability percentage for protocol processing and handling interrupts are $r/(\lambda + r)$ and $\lambda/(\lambda + r)$, respectively.

Thus, the mean effective service rate can be expressed as

$$\mu' = \mu \cdot \frac{r}{\lambda + r}. \tag{8}$$

Similar to analysis presented in Section 3.2 for determining the system throughput when not using DMA, this system can also be modeled as M/M/1/B queue as is the case for the ideal system. The system throughput can be expressed by equation (1). However, the mean service rate μ for equation (1) will have to be replaced by the mean effective service rate μ' of equation (8).

3.4 Special Case

We consider a special case when interrupt handling is ignored, i.e., when $T_{ISR} = 0$. In this situation when $T_{ISR} = 0$, $r \to \infty$ and $c \to \infty$. We prove that equations (5) and (6) give the original mean service rate μ as follows:

For mean effective service rate of equation (7),

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

For mean effective service rate of equation (8),

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

4 Numerical Examples

In this section, we report some numerical results of our analytical models to study the impact of interrupts on system throughput. The system throughput is studied as a function of packet arrival rate. A number of cases are considered: Ideal system, DMA enabled, and DMA disabled. In addition, our analytical results are plotted against experimental results reported by [3]. We use the same system parameter values as in [3]. More importantly, we show results when considering the two options of delivering packet data from kernel to user space. The first option is to have an extra copy and thus extending protocol processing time for each packet. The second option eliminates extra copying using pointer manipulations.

We first examine the system throughput employing the first option of having the network protocol processing copy the packet data from host kernel memory to user space. For this case and as measured by [3], the mean service time for ISR ($1/r$) is 95 μ seconds with DMA disabled and is 40 μ seconds with DMA enabled. The mean copy time from NIC to kernel memory ($1/c$) is 55 μ seconds with DMA disabled . The network protocol processing time ($1/\mu$), which includes copying of packet data to user space, is 150 μ seconds. We fix B to a size of 1000.

Figure 4 depicts the impact of high and low packet arrival rate on system throughput. It is observed that at low traffic rate, the system throughput is the same for network adaptors regardless whether these adaptors are DMA equipped or not. We note for the ideal system and when DMA is not utilized, the throughput is the expected one and matches closely the expected behavior in Figure 1. Also, the experimental results are very much inline with the analytical ones. As for the system throughput when utilizing DMA, we note that the throughput doesn't fall rapidly to zero but gradually decreases as the system load of packet arrivals increases.

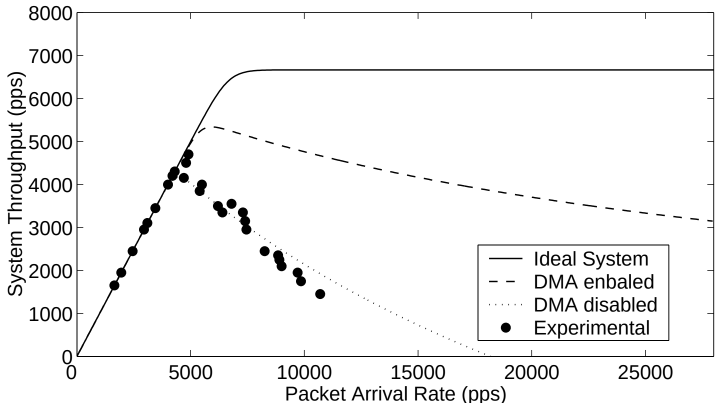

Figure 4. System throughput with copying from kernel to user space

We second examine the system throughput with the second option of not having the network protocol processing copy the packet data from host kernel memory to user space. This can be accomplished by manipulating pointers for both host kernel memory and user-space buffers used for user applications [5,6]. Similarly for this case, the mean service time for ISR ($1/r$) is 95 μ seconds with DMA disabled and is 40 μ seconds with DMA enabled. The mean copy time from NIC to kernel memory ($1/c$) is 55 μ seconds. However, the network protocol processing time ($1/\mu$) is reduced to 100 μ seconds. Again, we fix *B* to a size of 1000. Figure 5 depicts the impact of packet arrival rate on system throughput. Comparing results of Figure 5 with those of Figure 4, we note a noticeable improvement in the system throughput for all cases.

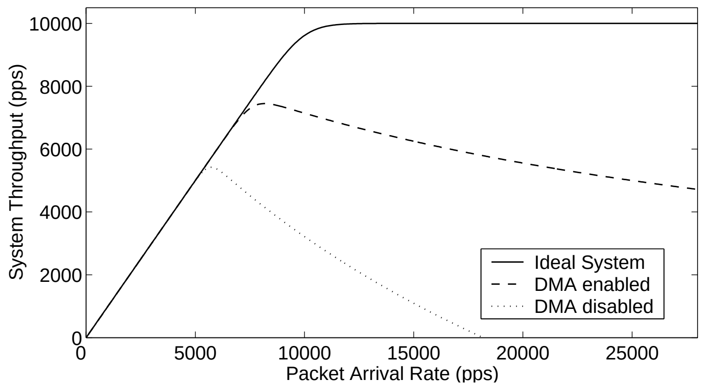

Figure 5. System throughput with no copying from kernel to user space

An important observation can be made about the receive-livelock point in which the system throughput reaches zero for systems where DMA is disabled. From Figure 4 and 5, it is observed that the receive-livelock still occurs at the same point regardless of improving the network protocol processing for packets. This observation is inline with our analytical solution. Based on equation (7), the receive-livelock point depends only the values of λ and c. It occurs precisely when $\lambda \geq c$. Hence for both figures, the receive-livelock point occurs when $\lambda = 1/c = 18{,}182$ pps. On the other hand when DMA is enabled, the receive-livelock point does not occur unless $\lambda \to \infty$, as illustrated by equation (8). Therefore, utilizing DMA engines for high-speed network adaptors becomes a very essential design and implementation consideration.

5 Validation of Analysis

In order to validate our analytical models, we built a discrete-event simulation using C programming and ran a wide number of simulation runs. In all cases, a perfect accordance has been verified based on the assumptions of analysis using exponential service times and Poisson arrivals. Results of analysis also gave a good approximation against results obtained by simulation runs that considered constant values for both ISR handling and protocol processing times. Due to the allowed space of this publication, a complete description of the simulation models will be presented in a future publication. In addition and as a verification, it was shown in Section 3.4 that when ignoring interrupt overhead, the mean effective service rates for the two models of DMA-equipped and DMA-not-equipped network adaptors turn out to be the same as the original mean service rate for the ideal system. More importantly and as shown in Section 4, our analytical results were comparable to experimental and measured results reported by [3]. In [3], traffic was generated back to back at full speed and with packets of fixed size.

6 Conclusion

We presented analytical models that capture the receive livelock phenomenon. Receive livelock occurs in interrupt-driven operating systems due to the high interrupt rate generated by packet arrivals of high-speed networks. The system throughput for both DMA-equipped and DMA-not-equipped network adaptors is studied. We concluded that at light network traffic, the system throughput is the same for network adaptors with or with no DMA engines. However at high network traffic, the system throughput is degraded significantly for those network adaptors not equipped with DMA engines. Utilizing DMA engines for high-speed network adaptors becomes very important design and implementation consideration in order to minimize the negative impact of interrupt overhead on system throughput. A noticeable decrease in system throughput was observed when performing multiple copies of received packet data. Analytical models were validated by simulation. Also reported experimental results show that our analytical models give good approximation. The impact of generating bursty traffic instead of Poisson is being studied by the authors using simulation, and results are expected to be reported in the near future. A lab experiment is also being set up to measure and compare the system performance for all cases. As a further work, we will study and evaluate the performance of the different proposed solutions for minimizing and eliminating interrupt overhead.

References

[1] I. Kim, J. Moon, and H. Y. Yeom, “Timer-Based Interrupt Mitigation for High Performance Packet Processing, ” *Proceedings of 5th International Conference on High-Performance Computing in the Asia-Pacific Region*, Gold Coast, Australia, September, 2001.

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

[3] J. Mogul, and K. Ramakrishnan, “Eliminating Receive Livelock In An Interrupt-Driven Kernel,” *ACM Trans. Computer Systems*, vol. 15, no. 3, August 1997, pp. 217-252.

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

[5] P. Druschel, and G. Banga, “Lazy Receive Processing (LRP): A Network Subsystem Architecture for Server Systems,” *Proceedings Second USENIX Symp. on Operating Systems Design and Implementation*, October 1996, pp. 261-276.

[6] 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.

[7] 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.

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