Modeling and Analysis of Interrupt Disable-Enable Scheme

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

Abstract

System performance of Gigabit network hosts can severely be degraded due to interrupt overhead caused by heavy incoming traffic. One of the most popular solutions to mitigate such overhead is interrupt disabling and then enabling. In this solution, interrupt overhead is significantly reduced by disabling interrupts and only re-enabling them after processing all queued packets. In this paper we investigate analytically the performance of the scheme of interrupt disabling and enabling and compare it with normal interruption and interrupt coalescing. The system performance is analyzed and compared in terms of throughput, latency, and CPU availability for user applications.

1. Introduction

With almost all today's Gigabit NICs, an incoming packet gets transferred (or DMA'd) through the PCI bus from the NIC to the protocol processing buffer of the kernel. After the packet has been successfully DMA'd, the NIC generates an interrupt to notify the kernel to start protocol processing of the incoming packet. Protocol processing typically involves TCP/IP processing of the incoming packet and delivering it to user applications. During protocol processing, other packets may arrive and get queued. Protocol processing time is affected by the interrupts of incoming packets. Interrupt handling has an absolute priority over protocol processing. If an interrupt occurs during protocol processing, the protocol processing will be disrupted or preempted (i.e., protocol processing will stall until the completion of interrupt handling). Therefore, the protocol processing time can be enormously stretched. If interrupt rate is high enough (as the case in Gigabit networks), the system will spend all of its time responding to interrupts, and hence the system throughput will drop to zero. This situation is called receive livelock [1]. In this situation, the system is not deadlocked but causing tasks scheduled at a lower priority to starve.

A number of solutions has been proposed in the literature to mitigate interrupt overhead and improve 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. One of the most popular solutions to mitigate the interrupt overhead for Gigabit network hosts is interrupt disabling and then enabling. The key idea behind interrupt disable-enable scheme is inspired by [1]. Lately such scheme has been utilized by some kernels such as the case of Linux NAPI [2]. The idea is to have the interrupts of incoming packets turned off or disabled as long as there are packets to be processed by the kernel's protocol stack, i.e., the protocol buffer is not empty. When the buffer is empty, the interrupts are turned on again or re-enabled. Any incoming packets (while the interrupts are disabled) are DMA'd silently to protocol buffer (called Rx DMA Ring in Linux 2.6) without incurring any interrupt overhead. Therefore, interrupt overhead is only associated with the first incoming packet that arrives when the buffer is empty. Subsequent incoming packets that get queued in the protocol buffer do not introduce interrupt overhead. Intuitively, such scheme is expected to mitigate a significant amount of interrupt overhead when compared to normal interruption (where every incoming packet incurs an interrupt overhead).

In previous work [3], we presented an analytical and simulation study of normal interruption. In [4], we presented a complete analytical model to study the performance of interrupt coalescing scheme. In this paper, we present a novel analytical study and models for the scheme of interrupt disabling and enabling. The performance of scheme of interrupt disable-enable is compared with that of normal interruption and interrupt coalescing. The performance is studied and compared in terms of three key performance indicators which include system throughput, system latency, and

CPU availability for other processing including user applications.

The rest of the paper is organized as follows. Section 2 summarizes previously related analytical work for three interrupt handling schemes: ideal, normal interruption, and interrupt coalescing. Section 3 presents an analytical model using Markov chains of the scheme of interrupt disable and enable. Section 4 compares the four interrupt handling schemes by giving numerical examples showing both analysis and simulation results. Finally, Section 5 concludes the study and identifies future work.

2. Related Analytical Work

We will summarize previously related analytical work for related analytical work for three interrupt handling schemes: ideal, normal interruption, and interrupt coalescing. Later in Section 4, we will compare the performance of these schemes with the scheme of interrupt disable and enable.

2.1 Ideal and Normal Interruption

In previous work [3], we presented analytical models to study two types of interrupt handling schemes (viz. ideal scheme and normal interruption). In ideal scheme, the overhead involved in generating interrupts is totally ignored. The ideal scheme gives the best performance that can possibly be obtained when employing interrupts, thus serving as a reference (or a benchmark) to compare with. In normal interruption, every incoming packet causes an interrupt. Closed-form solutions for a number of performance metrics can be found in [3].

2.2 Interrupt-Coalescing (IC)

One of the most popular solutions to mitigate interrupt overhead for Gigabit network hosts is interrupt coalescing or IC. In recent years most network adapters or NICs are manufactured to have interrupt coalescing. Additionally, many operating systems, including Windows 2000 and Linux, support IC. IC is a mode or a feature in which the NIC generates a single interrupt for a group of incoming packets. This is opposed to normal interruption mode in which the NIC generates an interrupt for every incoming packet. In interrupt-coalescing (IC) mode, there are two schemes to mitigate the rate of interrupts: count-based IC and time-based IC. In count-based IC mode, the NIC generates an interrupt when a predefined number of packets (denoted by τ) has been received. In time-based IC mode, the NIC waits a predefined time period (denoted by T) before it generates an interrupt. During this time period multiple packets can be received. The coalescing parameters of τ and T are tunable and configurable parameters which are set by the device driver. Analytical models and closed-form solutions for system throughput, latency, and CPU availability were given in [4]. The same underlying assumptions and notations used in this paper were used in [4].

3. Modeling and Analysis of Interrupt Disabling and Enabling

Under the scheme of interrupt disabling and enabling, an incoming packet (while the protocol processing buffer is empty) would cause an interrupt when it first arrives. The ISR first disables the interrupt and then notifies protocol processing to process the packet. Protocol processing will keep processing packets as long as the buffer is not empty. When the buffer is empty, protocol processing will re-enable the interrupts. During protocol processing, incoming packets will not introduce interrupt overhead and will get queued (or DMA'd) quietly onto the protocol buffer.

For comparison purposes, we will use the same assumptions and notations presented in [3]. We assume Poisson incoming traffic, fixed packet sizes, and exponential times for interrupt handling and protocol processing.

Let λ denote the mean incoming packet arrival rate,

μ denote the mean protocol processing rate carried out by the kernel, and thus $1/\mu$ becomes 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, and

$1/r$ denote the mean ISR or interrupt handling time (i.e., 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.

Figure 1 exhibits a Markov chain model for the behavior of the interrupt disable-enable scheme. The state space is defined as

$$S = \{(n, 0), 1 \le n \le B\} \cup \{(n, 1), 0 \le n \le B\}$$

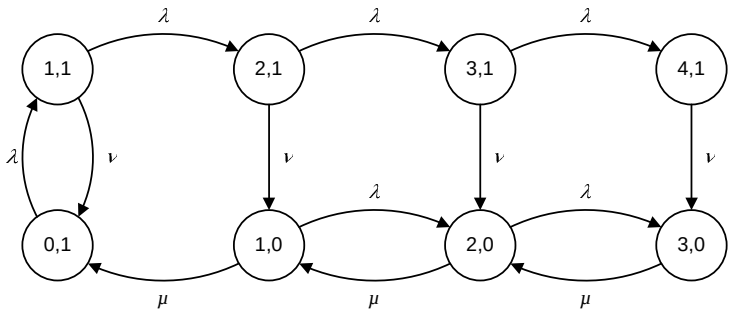


Figure 1. Markov state transition diagram for modeling interrupt disable-enable scheme

where n denotes the number of packets in the buffer, and B denotes the buffer size. States $(n,1)$ define the states in which the interrupts are enabled. States $(n,0)$ define the states in which the interrupts are disabled. State $(0,1)$ represents the state where the system is idle (with no packets) and the interrupts are enabled. We let $1/\nu$ denote the mean processing service time when the interrupts are enabled. We assume $1/\nu$ is exponentially distributed and it includes: the time to disable the interrupts, the time to handle interrupt (with a mean of $1/r$), and the time to service one packet by the kernel's protocol stack (with a mean of $1/\mu$). For simplicity, we ignore the time for re-enabling the interrupt. We believe it is very small and negligible (typically one or two write instructions to the NIC's control register).

Let $p_{n,m}$ be the steady-state probability at state (n,m). The stationary equations of the Markov chain in Figure 1 are:

$$0 = -\lambda\ p_{0,1} + \nu\ p_{1,1} + \mu\ p_{1,0} \quad (1a)$$

$$0 = -(\lambda+\nu)\ p_{1,1} + \lambda\ p_{0,1} \quad (1b)$$

$$0 = -(\lambda+\mu)\ p_{1,0} + \nu\ p_{2,1} + \mu\ p_{2,0} = 0 \quad (1c)$$

$$0 = -(\lambda+\nu)\ p_{n,1} + \lambda\ p_{n-1,1}, \quad n \geq 2 \quad (1d)$$

$$0 = -(\lambda+\mu)\ p_{n,0} + \lambda p_{n-1,0} + \nu\ p_{n+1,1} + \mu\ p_{n+1,0}, n \geq 2 \quad (1e)$$

Before we proceed to present the solution of the above system, we will use the abbreviations $a = \lambda/\mu$ and $b = \lambda/(\lambda+\nu)$. Now, by using equations (1b) and (1d), we have

$$p_{1,1} = \frac{\lambda}{\lambda+\nu}\, p_{0,1} = b\ p_{0,1}$$

$$p_{2,1} = \frac{\lambda}{\lambda+\nu}\, p_{1,1} = b^2 p_{0,1}$$

$$p_{3,1} = \frac{\lambda}{\lambda+\nu}\, p_{2,1} = b^3 p_{0,1}$$

And by using mathematical induction, we get

$$p_{n,1} = b^n p_{0,1} \qquad n \geq 1 \quad (2)$$

Similarly, equations (1a), (1c), and (1e) can be used iteratively to obtain the following:

$$p_{1,0} = \frac{\lambda}{\mu} p_{0,1} - \frac{\nu}{\mu} p_{1,1} \quad (3)$$

But $\frac{\nu}{\mu}$ can be expressed as $\frac{a-ab}{b}$, then substituting this expression into equation (3) we get

$$p_{1,0} = a\ p_{0,1} - (a-ab)p_{0,1} = ab\ p_{0,1}$$

$$\begin{aligned} p_{2,0} &= \left(\frac{\lambda+\mu}{\mu}\right)p_{1,0} - \frac{\nu}{\mu}\, p_{2,1} \\ &= ab(a+1)\, p_{0,1} - b(a-ab)\, p_{0,1} \\ &= (a^2b + ab^2)\, p_{0,1} \end{aligned}$$

$$\begin{aligned} p_{3,0} &= \left(\frac{\lambda+\mu}{\mu}\right)p_{2,0} - \frac{\lambda}{\mu}\, p_{1,0} - \frac{\nu}{\mu}\, p_{3,1} \\ &= (a+1)(a^2b + ab^2)\, p_{0,1} - a^2b\ p_{0,1} - b^2\,(a-ab)\, p_{0,1} \\ &= (a^3b + a^2b^2 + ab^3)\, p_{0,1} \end{aligned}$$

$$\begin{aligned} p_{4,0} &= \left(\frac{\lambda+\mu}{\mu}\right)p_{3,0} - \frac{\lambda}{\mu}\, p_{2,0} - \frac{\nu}{\mu}\, p_{4,1} \\ &= (a+1)(a^3b + a^2b^2 + ab^3)\, p_{0,1} - (a^3b + a^2\, b^2)\, p_{0,1} \\ &\quad - b^3(a-ab)\, p_{0,1} \\ &= (a^4b + a^3b^2 + a^2b^3 + ab^4)\, p_{0,1} \end{aligned}$$

At this point it is reasonable to conjecture that

$$p_{n,0} = (a^n b + a^{n-1}b^2 + \cdots + a^2 b^{n-1} + a b^n)\, p_{0,1} \qquad (4)$$

Equation (4) can be verified by using mathematical induction also. We assume that equation (4) holds for $n-1$ and n, and we want to show that it also holds for $n+1$ as follow. Using (1e) and (4) we get

$$\begin{aligned} p_{n+1} &= \left(\frac{\lambda+\mu}{\mu}\right) p_{n,0} - \frac{\lambda}{\mu} p_{n-1,0} - \frac{\nu}{\mu} p_{n+1,1} \\ &= (a+1)(a^n b + a^{n-1}b^2 + \cdots + a^2 b^{n-1} + a b^n)\, p_{0.1} \\ &\quad - a(a^{n-1}b + a^{n-2}b^2 + \cdots + a^2 b^{n-2} + a b^{n-1})\, p_{0,1} \\ &\quad - b^n (a - ab)\, p_{0,1} \\ &= (a^{n+1}b + a^n b^2 + \cdots + a^3 b^{n-1} + a^2 b^n)\, p_{0,1} \\ &\quad + (a^n b + a^{n-1}b^2 + \cdots + a^2 b^{n-1} + a b^n)\, p_{0,1} \\ &\quad - (a^n b + a^{n-1}b^2 + \cdots + a^3 b^{n-2} + a^2 b^{n-1})\, p_{0,1} \\ &\quad - a b^n p_{0,1} + a b^{n+1} p_{0,1} \\ &= (a^{n+1}b + a^n b^2 + \cdots + a^2 b^n + a b^{n+1})\, p_{0,1} \end{aligned}$$

We can rewrite equation (4) as follow:

$$\begin{aligned} p_{n,0} &= \sum_{i=1}^{n} a^i b^{n-i+1}\, p_{0,1} \\ &= b^{n+1} p_{0,1} \cdot \sum_{i=1}^{n} (a/b)^i = b^{n+1} \cdot \left(\frac{a/b - (a/b)^{n+1}}{1 - a/b}\right) \cdot p_{0,1} \\ &= \left(\frac{a b^{n+1} - a^{n+1} b}{b-a}\right) p_{0,1} = \frac{ab}{b-a}(b^n - a^n)\, p_{0,1} \end{aligned} \qquad (5)$$

To obtain $p_{0,1}$, we utilize the fact that $\sum p_{n,m} = 1$. Hence

$$\begin{aligned} 1 &= p_{0,1} + \sum_{n=1}^{\infty} p_{n,0} + \sum_{n=1}^{\infty} p_{n,1} \\ &= \left(1 + \frac{ab}{b-a}\sum_{n=1}^{\infty}(b^n - a^n) + \sum_{n=1}^{\infty} b^n\right) p_{0,1} \\ &= \left(1 + \left(\frac{ab}{b-a}\right)\left(\frac{b}{1-b} - \frac{a}{1-a}\right) + \frac{b}{1-b}\right) p_{0,1} \\ &= \frac{1 - a + ab}{(1-a)(1-b)}\, p_{0,1}. \end{aligned}$$

Therefore,

$$p_{0,1} = \frac{(1-a)(1-b)}{1-a+ab}. \qquad (6)$$

It is worth mentioning that $\sum_{n=1}^{\infty} b^n - a^n$ is a geometric series and it converges only if a and b are less than 1. The value of b is always less than 1. And therefore, the only condition for the existence of the steady-state solution is that a must be less than 1, or equivalently, $\lambda < \mu$.

CPU availability. CPU availability for user applications is basically the idleness state which can be given by equation (6).

Mean System Throughput. The mean system throughput γ can be expressed as

$$\begin{aligned} \gamma &= \sum_i \mu_i p_i \\ &= \sum_{n=1}^{\infty} \nu\, p_{n,1} + \sum_{n=1}^{\infty} \mu\, p_{n,0} \\ &= \left(\nu \sum_{n=1}^{\infty} b^n + \mu \frac{ab}{b-a} \sum_{n=1}^{\infty} (b^n - a^n)\right) p_{0,1}, \end{aligned}$$

which can be subsequently simplified to

$$\gamma = \frac{\nu(1-a) + \mu ab}{1 - a + ab}. \qquad (7)$$

Mean System Latency. The mean system latency $E(T)$, can be expressed straightforward as

$$E(T) = \frac{E(n)}{\lambda}, \qquad (8)$$

where $E(n)$ is the average number of packets in protocol buffer and can be expressed as

$$\begin{aligned} E(n) &= \sum_{n=1}^{\infty} n\, p_{n,0} + \sum_{n=1}^{\infty} n\, p_{n,1} \\ &= \left(\sum_{n=1}^{\infty} n b^n + \frac{ab}{b-a} \sum_{n=1}^{\infty} n (b^n - a^n)\right) p_{0,1} \\ &= \left(\frac{b}{(1-b)^2} + \frac{ab}{b-a} \cdot \left(\frac{b}{(1-b)^2} - \frac{a}{(1-a)^2}\right)\right) p_{0,1} \\ &= \left(\frac{b(1-a) + a^2 b(1-b)}{(1-a)^2 \cdot (1-b)^2}\right) p_{0,1} \end{aligned}$$

Thus,

$$E(n) == \frac{b(1-a) + a^2 b(1-b)}{(1-a)(1-b)(1-a+ab)}. \qquad (9)$$

Special Case. Let us consider the special case when $\nu = \mu$. It can be easily verified that all the equations derived for the throughput, latency and CPU availability will be exactly reduced to those of a pure

M/M/1 queueing system with an arrival rate of λ and service rate of μ.

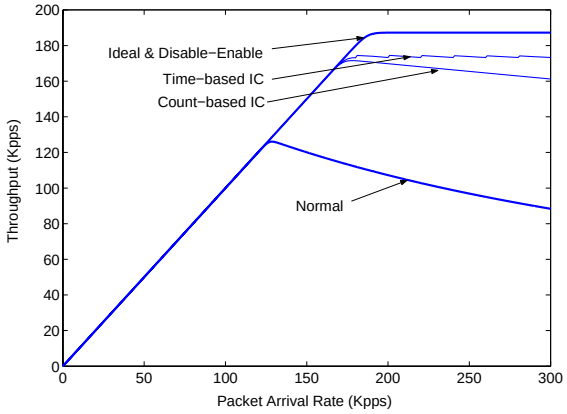
4. Simulation.

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 [5]. A detailed description and flowcharts of the simulation model for normal interruption can be found in [3]. The simulation model reported in [3] was modified for the scheme of interrupt disabling and enabling.

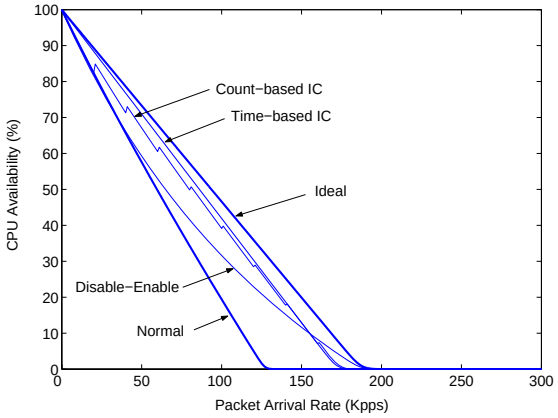
5. Numerical Examples

In this section, the results of analysis and simulation are reported. Numerical results are given for key performance indicators. These indicators include mean system throughput, CPU availability, latency, and packet loss. We compare the performance for the ideal system, normal interruption, and IC. We use a mean interrupt handling time ($1/r$) of 3.73 μs and a protocol processing ($1/\mu$) of 5.34 μs. For interrupt disabling-enabling scheme, there is also the incurred cost of writing to the NIC control registers to disable and enable interrupts of incoming packets. We assume the cost of writing to the NIC register is 0.5 μs. As a consequence, the parameter $1/v$ in analysis, which denotes the mean protocol processing service time when interrupts are enabled, is approximately equal to 9.57 μs, that is 0.5 + 3.73 + 5.34 μs. In all of our examples, a kernel's protocol processing buffer B of a size of 1000 packets is used. These values are realistic and selected based on experimental findings reported in [6,7].

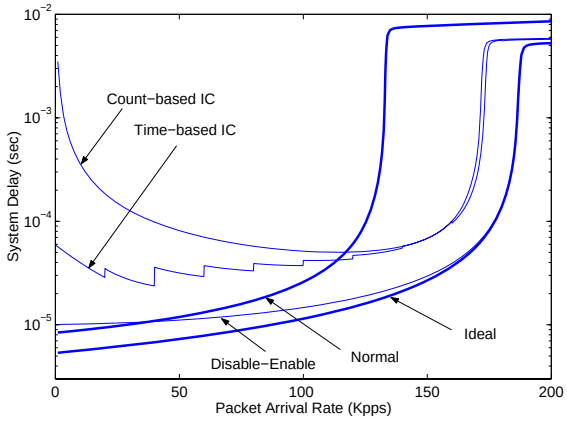
Figure 2 plots the mean system throughput, CPU availability for user applications, and mean system latency as a function of the system load represented by the packet arrival rate. The load and throughput are both expressed in pps (packets per second). Analysis and simulation results were in very close agreement. From the figures, it is observed that the maximum throughput occurs at 187 Kpps. For normal interruption, it can be noted that the saturation or cliff point for the system occurs at 127 Kpps. At this point, the corresponding CPU utilization (for ISR handling plus protocol processing) is at 100%, and thus resulting in a CPU availability of zero. Therefore, user applications will starve and livelock at this point. Figure 2(a) shows that as the arrival rate increases after the cliff point the system throughput starts to decline.



(a)



(b)



(c)

Figure 2. Key performance indicators in relation to traffic load

One observation can be made about IC schemes with a parameter of $\tau = 1$ in case of count-based coalescing and $T = 0$ in case of time-based coalescing. It is observed that in such cases, both coalescing scheme resort exactly, as expected, to normal interruption. Also from the figure, it is depicted that the analysis curves for time-based coalescing (more noticeable in Figure 7(b) and 7(c) at very low rate) are not smooth. As illustrated in [4], the analysis for time-based coalescing is performed based on the analysis of count-based coalescing with the coalescing parameter τ being an integer and approximated to $\lceil \lambda T \rceil$. Thus, τ takes on discrete values and remains unchanged until a different value is produced as λ changes in $\lceil \lambda T \rceil$.

There are also a number of important observations and conclusions to be made when examining and comparing the performance of all interrupt handling schemes. It can be concluded that no single scheme gives the best performance. For example, the scheme of interrupt disabling and enabling outperforms all other schemes in terms of throughput and latency. However in terms of CPU availability, the interrupt disabling and enabling gives the worst performance second to normal interruption. Also at extremely low rate interrupt disabling and enabling gives worse latency than normal interruption.

6. Concluding Remarks

We developed an analytical model to analyze and study the performance of Gigabit-network hosts when employing the interrupt handling scheme of disabling and enabling for the mitigation of interrupt overhead caused by incoming traffic. The analytical model was verified and validated by simulation and by considering special cases. The performance was studied in terms of system throughput, CPU availability, and latency. It was concluded that no particular interrupt handling scheme gives the best performance under all load conditions. Selection of the most appropriate scheme to employ depends primarily on system performance requirements, user's most important performance metric, and present traffic load. It was demonstrated by giving numerical examples that the scheme of disabling and enabling interrupts outperforms, in general, all other schemes in terms of throughput and latency. However when it comes to CPU availability, the scheme of interrupt coalescing is more appropriate. Polling is yet another potential scheme to mitigate interrupt overhead. Polling is designed to primarily solve the issue of CPU availability under high load. However, polling is expected to perform poorly under low load. A hybrid scheme of interrupt disable-enable and polling or coalescing can be an attractive solution. As a further study, we are in the process of studying the performance of polling. Performance and comparison results are to be published in the near future.

Acknowledgements

We acknowledge the support of King Fahd University of Petroleum and Minerals in completion of this work. This work has been funded under Project #FT-2005/17.

7. References

[1] 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.

[2] J. H. Salim, "Beyond Softnet," Proceedings of the 5th Annual Linux Showcase and Conference, November 2001, pp 165-172

[3] K. Salah and K. El-Badawi, "Analysis and Simulation of Interrupt Overhead Impact on OS Throughput in High-Speed Networks," International Journal of Communication Systems, vol. 18, no. 5, Wiley Publisher, June 2005, pp. 501-526

[4] K. Salah "To Coalesce or Not to Coalesce", International Journal of Electronics and Communications (AEU), (In Press.)

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

[6] M. Aron and P. Druschel, "Soft Timers: Efficient Microsecond Software Timer Support for Network Processing," ACM Transactions on Computer Systems, vol. 18, no. 3, August 2000, pp. 197-228.

[7] Ashford Computer Consulting Service, "GigaBit Ethernet to the Desktop – Client1 System Benchmarks", 2004, http://www.accs.com/p_and_p/GigaBit/Client1.html