Journal of Applied Sciences

ISSN 1812-5654

science alert

ANSInet
an open access publisher
http://ansinet.com

Journal of Applied Sciences 6 (6): 1396-1403, 2006
ISSN 1812-5654
© 2006 Asian Network for Scientific Information

Performance Evaluation of Various Techniques for Handling Interrupts in Gigabit Networking Environment

Salman A. AlQahtani and Khalid Salah
Computer Centre, King Fahd Security College, Riyadh, Saudi Arabia

Abstract: Multimedia applications over high-speed networks can generate heavy load conditions. When the network end system is involved in processing this high network traffic, its performance depends critically on how its tasks are scheduled. The polices and mechanism that schedule incoming network traffic and other tasks should guarantee acceptable system throughput, reasonable latency and jitter (variance in delay), reasonable system availability, fair allocation of CPU resources among multimedia traffic reception, packets transmission, protocol processing, application processing and over all system stability, without imposing excessive overhead, especially in case of high traffic load. In this study we compare different schemes for handling interrupts in Gigabit networking environment (high-speed network interface) then we compare the performance measures of hard timer and soft timer polling schemes used for high-speed network interface with high traffic load. In addition to the throughput performance metric in which most of the literatures focus on only, other performance metrics such as CPU availability, loss ratio, packet delay are defined and studied. A simulation is used to study the impact of interrupt overhead caused by high-speed network traffic on operating system (OS) performance. The performance evaluations, which are performed using a discrete event simulation, indicate that under conditions of high traffic load, the polling system offers increased throughput and reduced latency for traffics.

Key words: High-speed networks, NIC, interrupts, operating system, performance evaluation

INTRODUCTION

The explosive growth of the high-speed multimedia networks and the widespread use of web-based related applications place new demands on the network end system such as PC-Router, network server and host connected to high speed links. Multimedia applications over high-speed networks can generate heavy load conditions. When the network end system is involved in processing this high network traffic, its performance depends critically on how its tasks are scheduled. The polices and mechanism that schedule incoming network traffic and other tasks should guarantee acceptable system throughput, reasonable latency and jitter (variance in delay), reasonable system availability, fair allocation of CPU resources among multimedia traffic reception, packets transmission, protocol processing, application processing and over all system stability, without imposing excessive overhead, especially in case of high traffic load.

We can define throughput as the rate at which the system delivers packets to their ultimate consumers. A consumer could be an application running on the receiving network end system, or the network end system could be acting as a router and forwarding packets to consumers on other hosts. We expect the throughput of a well-designed system to keep up with the offered load up to a point called the Maximum Loss Free Receive Rate (MLFRR) and at higher loads throughput should not drop below this rate. Such rate is an acceptable rate and is relatively flat after that. Of course, useful throughput depends not only on successful reception of packets but also the system must transmit the packets. Multimedia applications often depend more on low-latency and low-jitter communications than on high throughput. During high traffic loads, we want to avoid long queues, which increase latency and bursty scheduling, which increases jitter. When the network end system is overloaded with incoming network packets, it must also continue to process other tasks, so as to keep the system responsive to the management and control requests and to allow applications to make use of the arriving packets. Availability is the percentage quantity that measures how much of the time the CPU power is available for other processes including user applications. This is actually the probability when there is no polling processing and there are no packet being processed by the protocol stack.

Corresponding Author: Salman A. AlQahtani, Department of Computer Engineering, King Fahd University of Petroleum and Minerals Mail Box 5065, Dhahran 31261, Saudi Arabia Tel: 996 3 860 2110, 966505225172

Therefore, the scheduling subsystem must fairly allocate CPU resources among packet reception, packet transmission, protocol and application processing.

The mean contributions of this study are two-fold. First, you compare various techniques for handling interrupts in Gigabit networking environment (high-speed network interface). Second, we compare the performance measures of hard timer and soft timer polling schemes used for high-speed network interface under high traffic load. Most of the previous works study the system performance from the throughput point of view only. In addition to the throughput performance metric, other performance metrics such as CPU availability, loss ratio, packet delay are defined and studied. Therefore, our objective is to produce a reliable, low latency and acceptable throughput, Network Interface Card (NIC) scheduling scheme for Gigabit Ethernet. The studied mechanisms are evaluated and compared using a discrete event simulation under high traffic load. A simulation is used to study the impact of interrupt overhead caused by high-speed network traffic on operating system (OS) performance.

NETWORK INTERFACE DESCRIPTION AND MODEL

The network interface consists of several hardware and software interacting units in both the computer and the NIC. The NIC consists of a receive part and a transmit part which are completely symmetrical (Ramakrishnan, 1993). The major components of the receive part of the network interface system are shown in Fig. 1. We generally focus on the performance of the receive side of the network interface and attempt to ensure that it is designed well. The design of the receive side is critical because there is less control over receiving concurrently multiple-traffic from multiple stations and the bursty nature of the received traffic.

The Programmed Input/Output (PIO) and Direct Memory Access (DMA) are two mechanisms available by which data can be moved from NIC to the host memory, or the reverse, as defined by the Peripheral Component Interconnect (PCI) specification. With PIO, the copying of an arrived packet from NIC buffer to host kernel memory or transferring of packets from host kernel memory to adapter is performed by the CPU as part of Interrupt Service Routine (ISR) handling for each incoming packet. A major drawback for a PIO-based design is burdening the CPU with copying incoming packets from the NIC to kernel memory. Currently, most network interfaces are DMA-capable. With DMA, the NIC directly reads and writes from/to the host system memory without any CPU

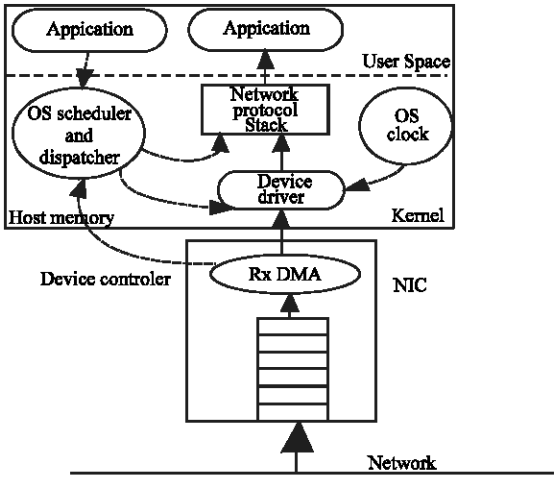

Fig. 1: The basic component of the receive part of the network interface

involvements. The CPU simply gives the NIC a memory address and the NIC writes to (reads from) it, through the bus interface such as the PCI. The CPU is therefore responsible for providing the NIC a pair of buffer descriptor lists; one to transmit out of and one to receive into. A buffer descriptor list is an array of address/length pairs (Ramakrishnan, 1993).

We consider here that all the network functionalities are performed by Operating System (OS) processes running in the kernel address space, while the application processes run in the user address space. When packets arrive at the receive part, the DMA engine handles the movement of packets from the NIC internal buffer to the host memory transparently. The NIC can read and write host memory without the need to buffer frames on the NIC internal buffer except view bytes of buffering to stage data between the bus and the link. After additional context switch, the device driver complete the receive transaction with the device controller and the packet processing continues with the network protocol processes (e.g., IP, UDP). Finally, the packet is copied from the kernel space to the user space and the recipient application is notified. 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 across the bus (Rizzo, 2002). In order to save CPU cycles consumed in copying packets, major network vendors equip high-speed NICs with DMA engines. The simple DMA-based architecture, shown in Fig. 1, implements all NIC packet transfer based on First Come First Serve (FCFS). The system model is disrobed in Fig. 2.

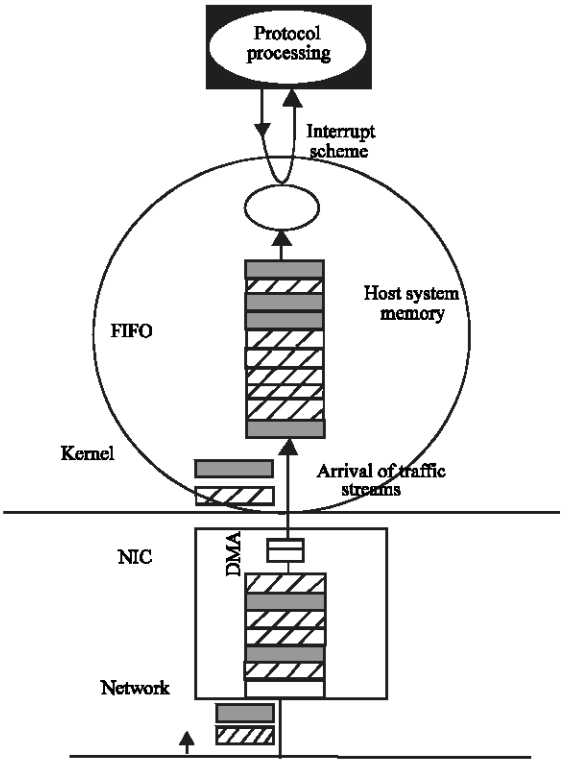

Fig. 2: System model

THE INTERRUPT HANDLING SCHEMES

Ideal mode: Interrupt overhead is totally ignored. In the ideal system, we assume the overhead involved in generating interrupts is totally ignored. The total service time includes only the time of processing the arrived packets.

Normal interruption: Every incoming packet generates an interrupt. If an interrupt is generated during Interrupt Service Routine (ISR), only one interrupt overhead is counted. ISR handling has a higher priority and preempts protocol processing. The total service time includes both the time of processing the arrived packets plus the interrupt overhead.

Disabling and enabling technique: The idea of pure interrupt disable-enable scheme is to have the interrupts turned off or disabled as long as there are packets to be processes by 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 quietly to protocol buffer without incurring any interrupt overhead. Note that at very low rate, each packet will experience approximately extra overhead to disable and re-enable (involving I/O write).

The polling scheme: In high-speed multimedia network interface, an alternative to interrupts is polling. The idea of polling is to disable interrupts of incoming packets altogether and thus eliminating interrupt overhead completely. In polling, the OS periodically polls its host system memory (i.e., protocol processing buffer) to find packets to process. The actual polling overhead is the cost of reading a status register on the host memory to check for a packet arrival and if one is detected to transfer the packet (s) from the host memory (i.e., protocol processing buffer) to the upper application through network protocol processing. Polling is used in systems that have a heavy network load, such as routers and bridges, firewalls, or file servers (Mogul and Ramakrishnan, 1997).

The main constrain that the high traffic network load imposes is that the polling period has to have as fine a time granularity as possible so that the packet latency to be minimized. There are two types of timers' facility, hard and soft timers. The hard timer is conventional timer facilities schedules events by invoking designated handler periodically in the context of hardware interrupt. In this case, the poll interval has fixed value (ex, 20 μs). The soft timer is an OS facility that allows efficient scheduling of software events at (μs) granularity. It is also possible to avoid substantial context switching overhead and cache pollution in polling by utilizing soft timers (Aron and Druschel, 2000). The basic idea behind soft timers is to take advantage of certain states (called OS trigger states) in the execution of a system where an event handler can be invoked (to poll the protocol buffer) at low cost. OS trigger states can occur when the system is already in the right context and has suffered cache pollution (e.g., at the end of handling a timer interrupt or a trap, when about to schedule an event or a task, at the end of a system call, or when CPU is executing idle loop). In addition, soft timer allows the dynamic adjustments of the poll interval. A drawback of soft timers is that they can only schedule events probabilistically (Aron and Druschel, 2000). In fact, using hardware timers to back up soft-timers, which what we actually do in this paper; allow very tight upper bounds on soft-timers delay at low costs.

In general, pure polling is rarely implemented. Polling with quota is usually the case whereby only a maximum number of packets are processed in each poll in order to leave some CPU power for application processing (Mogul and Ramakrishnan, 1997). 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, but this will not happen in our case where we assume high traffic loads. Second, processing of incoming packets is not performed immediately as the packets get queued until they are polled. However, this will be at the

benefit of saving some CPU time to upper applications. At each poll arrival, a weighted round-ribbon scheduling is implemented to select the next queue to be served.

System assumption and limitations: The proposed model is limited to the receive part of NIC equipped with DMA engines, where the interrupt overhead is more important. In addition, our design is restricted to a system with single processor. The system is studied under heavy traffic load because of high-speed multimedia networks. Therefore, the performance regarding the low traffic case will not be considered. Also due to high traffic, at each poll arrival, there is a packet to serve in the queue. At low load, 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, but this will not happen in our case due to high traffic loads. We avoid long queue, which increase the latency and bursty scheduling which increases jitter.

PERFORMANCE EVALUATIONS

Performance metrics: We study the system performance in terms of system throughput, CPU availability, loss rate (or blocking probability) and delay. The throughput is the total rate with which the application can read packets from the NIC. The Latency can be defined as the time interval between the arrivals of a packet to the network end system until its successful reach to the consumer. Availability is the percentage quantity that measures how much of the time the CPU power is available for other processes including user's applications. The loss ratio is define as the probabilities that the arrived packets are dropped due to space unavailability (buffer is full). Saturation point is the point at which the system cannot keep up with the offered load. These measures are evaluated when the simulation run ends as:

- System throughput = Σ packets processed/ simulation time;
- System availability = 1-(Σ packets processing time + Σ polling overhead time)/ simulation time;
- Delay time of packet = departure time - arrival time;
- System average latency of traffic = Σ packets delay times/ Σ number of packets processed;
- Blocking probability of traffic = Σ dropped packets/ Σ number of packets arrived;

Simulation parameters and assumptions: In this section, we define the system parameters. The parameters values and other assumptions are extrapolated form conducted experiments and studies appeared by Mogul and Ramakrishnan (1997), Morris *et al.* (2000) and Salah and Badawi (2003). The mean protocol processing rate carried out by the kernel is the time the system takes to process the incoming packet and delivers it to the application process. This time includes primarily the network protocol stack processing carried out by the kernel, excluding any interrupt handling. The polling period is the time between each consecutive poll event and the polling overhead time is the cost of reading a status register on the host memory to check for a packet arrival. Throughout our studies, we assume the following:

- Both of service times, protocol processing or Context Switching (CS) handling, change due to various system activities (Salah and Badawi, 2003; Morris *et al.*, 2000). Therefore, for our analysis, we assume both of these service times to be exponential.
- The network traffic follows a constant rate: we study the constant rate because it is closer in reality to the traffic generation characteristics used by the experiments by Mogul and Ramakrishnan (1997) Morris *et al.* (2000) as packets are generated back-to-back with almost a constant rate.
- The packet sizes are fixed. This assumption is true for Constant Bit Rate (CBR) traffic such as uncompressed interactive audio and video conferencing. Similarly, in (Morris *et al.*, 2000) traffic of fixed-size packets was generated back-to-back to the router. The packet's size for data can range from 64 to 1500 bytes.
- The queue size has a finite size.
- Hardware polling period is fixed where as software polling period has empirical distribution.
- Each poll period has a fixed quota of packets to be served defined as Quota.

A discrete-event simulation is developed and implemented by C programming language. The simulation follows closely and carefully the guidelines given by Law and Kelton (2000). Our simulations parameters values are based on what is reported by experiments by Mogul and Ramakrishnan (1997), Salah and Badawi (2003), Morris *et al.* (2000) and Aron and Druschel (2000). Hard timer polling period is 20 μs while software timer has empirical distribution with minimum 2 μs and bounded by 20 μs (see section 4). The polling overhead is 1.59 and 1.11 μs for hard and soft timers, respectively. The quotas used are 1, 2, 3, 4 packets per poll events. The service time rate is 5.6 μs and the per-packet overhead is (1/748) μs. The size of each queue is 500 packets, that is, 1000 packets in total. The network link speed is assumed to be 1 Gbps and the network traffic load varies.

SIMULATION RESULTS

The simulation results consist of two main parts. First part is to compare the performance of different interrupt handling and the second part is to study the hard-timer polling and soft-timer polling schemes using different quota limits. At each part's experiment the sources PCS gradually increase their aggregate offer loads from 20000 to 500,000 packets/sec while the systems (PC-router, Server, etc.) attempt to forward the packets as best as it can.

Interrupt schemes comparison: Here, we are to compare the performance of : 1) normal mode, 2) ISR, 3) Enabling Disabling interrupt (ISR-ED) and 4) Polling schemes. The system has one common queue of size 1000 packets for all packet types where no differentiation is made. System throughput, latency, CPU availability and system blocking probabilities are shown in Fig. 3-6, respectively. As seen in these figures in case of ISR handling, the throughput start decreasing at high traffic load leading to what is called livelock, because the CPU spent most of its time serving the interrupt. Therefore the system delay increases as high traffic increases (Fig. 5) and the CPU availability (Fig. 4) with respect to protocol processing reaches zero (CPU always busy serving the interrupt). Normal mode has the better performance at low traffic, but at high traffic ISR-ED and normal mode have the best performance in terms of system throughput, system delay and system locking probabilities. This positive performance is due to zero overhead encountered. But the problem is that, at high load the CPU availability is zero and this will prevent the higher application from being served and results in system bottlenecks (Fig. 4). The polling system is the intermediate solution where it has better performance than ISR and better CPU availability than normal and ISR-ED schemes. Moreover soft polling (STP) is better than hard polling (HTP) due low overhead in STP.

Polling schemes with different quota limits: In this subsection, we are to compare the performance of hard-timer polling and soft-timer polling schemes using different quota limits. For hard-timer polling, we use constant polling period of 20 μs and polling overhead of 1.59 μs. For soft-timer polling, we generate the measured empirical period Cumulative Distribution Function (CDF) of ST-Apache (Aron and Druschel, 2000) with a bound of 20 μs. The measured empirical period CDF of ST-Apache by Aron and Druschel (2000) gives us the resulting distribution of soft timers delay from 2 μs until 150 μs. As we state before, using hardware timers to back up soft-timers allow very tight upper bounds on soft-timers delay at low costs. Therefore, the new distribution of soft timer delay after bounding it by the hard timer (20 μs) is obtained by using the Best Fit package found. Figure 7

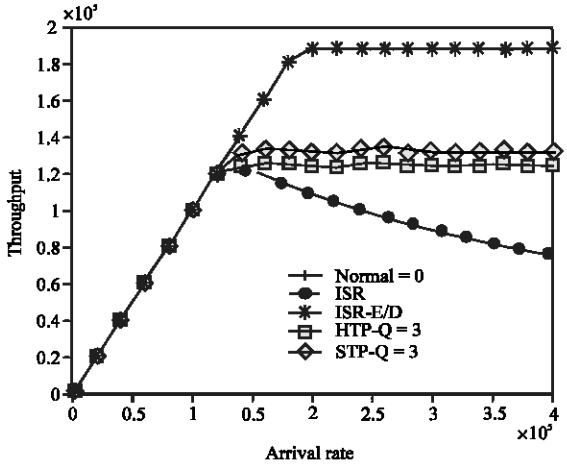

Fig. 3: System throughput

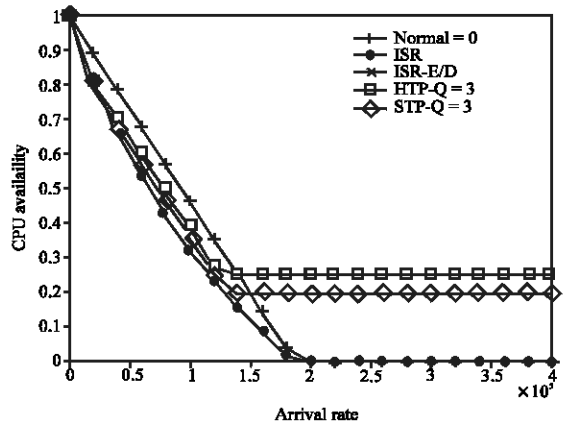

Fig. 4: CPU Availability

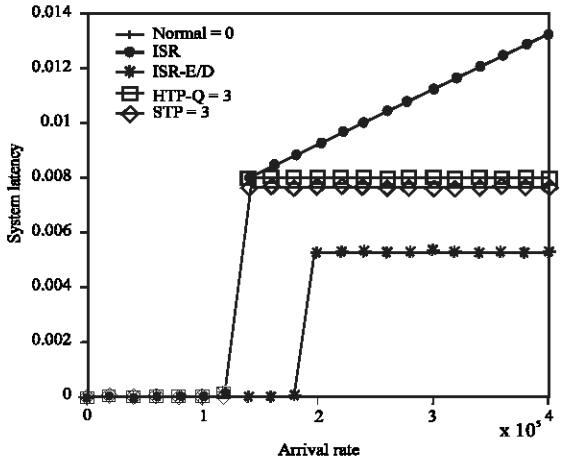

Fig. 5: System latency

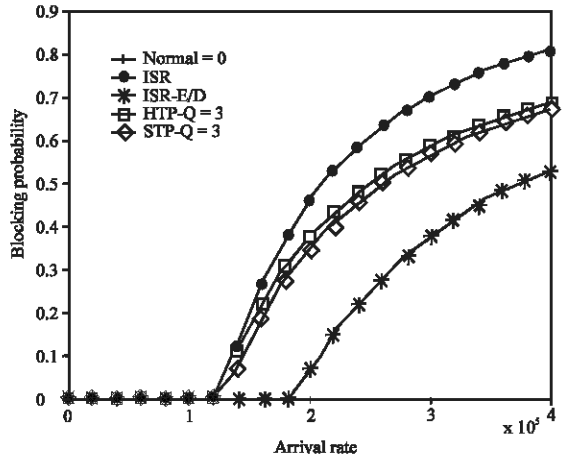

Fig. 6: System blocking probability

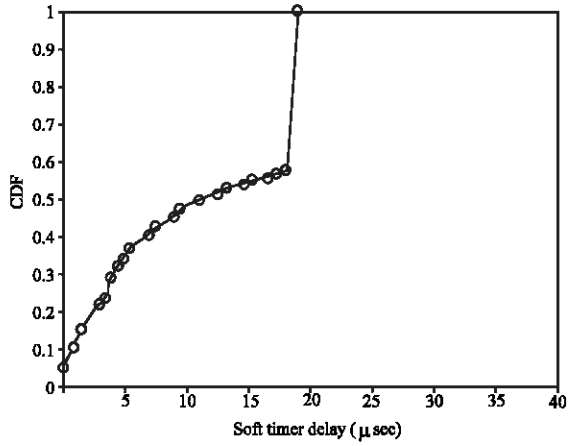

Fig. 7: Soft timer delay distribution

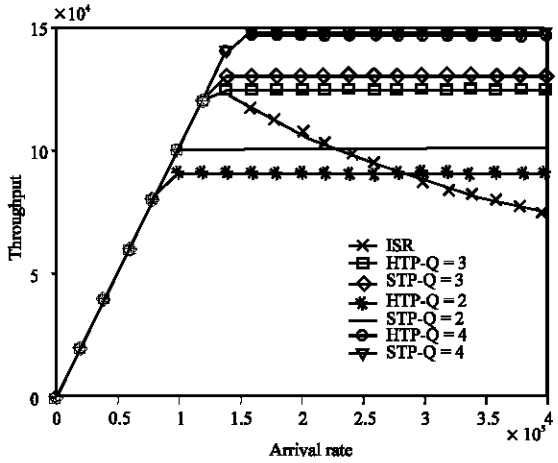

Fig. 8: System throughput

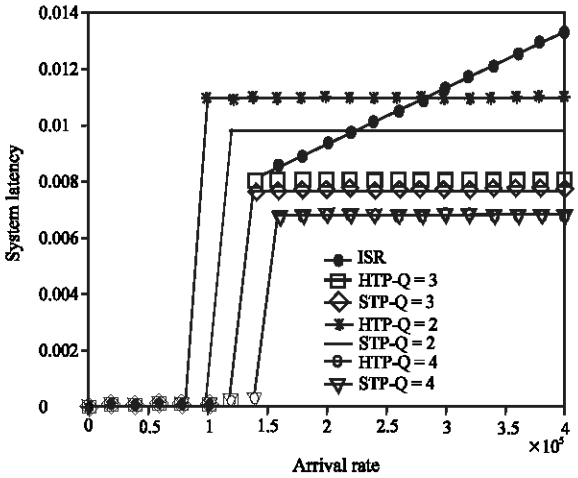

Fig. 9: System latency

shows our soft timer interval distribution, which is used to generate the soft timer poll period in our simulation. A polling overhead of 1.11 μs for soft-time polling is considered.

Four different quota limits is used to compare the system performance. The system has one common queue of size 1000 packets for all packet types where no differentiation is made. System throughput, latency and CPU availability are shown in Fig. 8-10, respectively. As seen in these Fig. 8-10 in case of ISR handling, the throughput start decreasing at high traffic load leading to what is called livelock, because the CPU spent most of its time serving the interrupt. Therefore the system delay increases as high traffic increases (Fig. 9) and the CPU availability (Fig. 10) with respect to protocol processing reaches zero (CPU always busy serving the interrupt). Using polling schemes, the system throughput keeps up with the offered traffic to MLFR rate at high traffic load (Fig. 8). In addition, the system delay and CPU availability keep with a fixed value at higher traffic (Fig. 9 and 10). For both soft and hard timer polling as we increase the quota limit the throughput has higher MLFR rate. At high quota (Q = 4) the throughput of soft timer and hard timer polling is almost equal in term of system throughput, delay and CPU availability and this is because that each poll period spent most of its time serving packets. But at low quota (1, 2, 3) the throughput of soft timer polling is higher than that of hard timer and this is because that the average poll period of soft timer is less than that of hard timers, therefore the buffer will poll more frequently than the hard polling. This will lead to lesser CPU availability and lesser delay than hard timer polling does. Moreover, Fig. 11 shows the blocking probability. As noticed, increasing the quota limit decrease the blocking probability and soft polling has lesser blocking probability.

To summarize, with very small quota with respect to the poll period, the throughput is very low and the latency

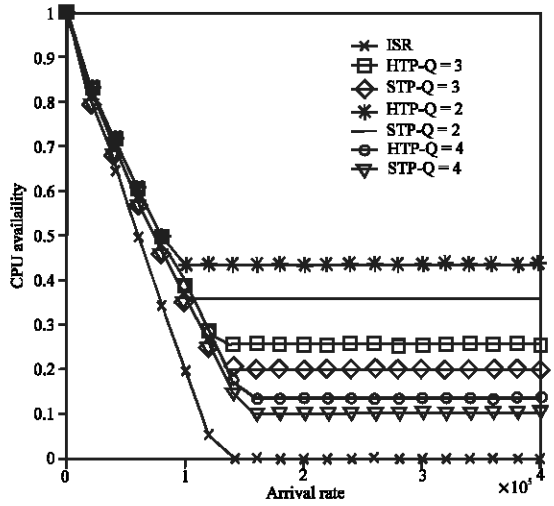

Fig. 10: CPU Availability

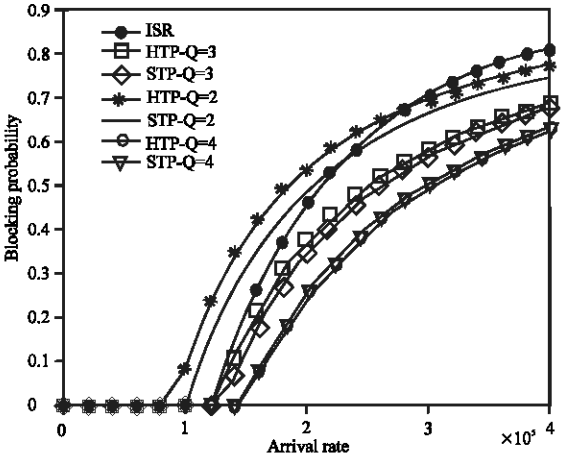

Fig. 11: System blocking probability

and CPU availability are very high, where as with very big quota with respect to the poll period, we will have higher throughput, lower latency and lower CPU availability. Therefore, selecting the quota is an important issue and based on the throughput, delay, CPU availability requirements, the required quota can be selected. A soft timer polling outperforms the hard timer with respect to the system throughput and latency, but the CPU availability is better in the case of hard timer polling. However, at very high quota this different disappears. In general, at higher quota packets, the hard timer polling is preferred since it gives reasonable throughput, latency and higher CPU availability to increase the packet forwarding to its higher applications.

RELATED WORKS

A number of solutions have been proposed in the literature to address network and system overhead and to 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. Interrupt overhead of Gigabit network devices can have a significant negative impact on system performance (Mogul and Ramakrishnan, 1997). Interrupt-driven systems can provide low overhead and good latency at low offered low, but degrade significantly at higher overload as in the high-speed network case (Mogul and Ramakrishnan, 1997). This will cause a receive livelock, in which the system spends all its time processing interrupts. The idea of polling is to disable interrupts of incoming packets altogether and thus eliminating interrupt overhead completely (Rizzo, 2002). Mogul and Ramakrishnan (1997) implemented a mechanism where interrupts are only used at low network load conditions, while in the high loads the interrupts are disabled and a polling thread is scheduled for reading the network interface. Every time a poll is executed, a certain packet quota is served. If at the end of polling some packets still remain at the NIC, the polling thread is executed again after a few milliseconds. Otherwise, the system switches back to interrupts. Hybrid Interrupt-Polling (HIP) scheme for the network interface exploits the trade-off between decreased receive-overhead and increased receive-latency (Dovrolis *et al*., 2001). This differs than (Mogul and Ramakrishnan, 1997), by adjusting the polling period based on the observed packets inter-arrivals.

One of the most popular solutions to mitigate interrupt overhead for Gigabit network hosts is interrupt coalescing (IC) (Zec *et al*., 2000; Druschel *et al*., 2000). Sometimes this scheme is known as interrupt batching in the literature, 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. Shivan *et al*. (1994) proposed Ethernet Message Passing (EMP), a completely new zero-copy, OS-bypass messaging layer for Gigabit Ethernet on Alteon NICs where the entire protocol processing is done at the NIC. OS bypass means that the OS is removed from the critical path of the message. Data can be sent/received directly into/from the application without intermediate buffering; making it true zero-copy architecture. The protocol has support for retransmission and flow control and hence is reliable. All parts of the messaging system are implemented on the NIC, including message-based descriptor management, packetization and reliability. The NIC handles all queuing, framing and reliability details asynchronously, freeing the host to perform useful work. Finally, the idea of using jumbo frames in Gigabits

network is introduced in (BestFit). In heavily loaded networks where continuous data transfer is required, current Ethernet frame size can actually degrade performance. Extended frames significantly enhance the efficiency of Ethernet servers and networks by reducing host packet processing by the CPU and increasing end-to-end throughput. Most of those solutions study the system performance from the throughput point of view. Other system performance such as, CPU availability, packet latency and blocking probabilities are defined and studies in this paper.

CONCLUSIONS

With the appearance of high-speed networks with link speed reaching gigabits per second, the network interfaces become the bottleneck of communication. In this study, we have compared various interrupt mechanisms design for high speed network interfacing. These mechanisms enable the NIC to efficiently support both high throughput and latency-critical applications, such as multimedia traffics. The idea of soft and hard timer is presented and used. Our discrete event-driven simulation model can be used as sold base to study other system performances metrics.

Performance evaluations indicate that under conditions of high traffic load, the proposed system offers increased throughput, stable overload behavior and reduced latency for traffics. The careful selection of the system parameters such as quota limit of each traffic class, polling period and queue size is an important issue. This selection allows the system designer to set the trade-off between them to the appropriate operating point in order to guarantee specific QoS requirements such as, CPU availability, delay, throughput and blocking probability. Through the event-driven simulation, we showed that the polling schemes are very efficient in case of high traffic streams. The idea of dynamically assigning the queue size, quota limit and the existing of more than one CPU in the NIC, are something that we plan to do in the future.

REFERENCES

Alteon WebSystems Inc., "Jumbo Frames," http://www.alteonwebsystems.com/products/white_papers/jumbo.htm.

Aron, M. and P. Druschel, 2000. Soft Timers: Efficient microsecond software time support for network processing. ACM Transactions on Computer Systems, 18: 197-228.

BestFit;http://www.palisade-europe.com/html/bestfit.html.

Dovrolis, C., B. Thayer and P. Ramanathan, 2001. HIP: Hybrid Interrupt-Polling for the Network Interface. ACM Operating Systems Reviews, 35: 50-60.

Druschel, P., L. Peterson and B.S. Davie, 1994. Experiences with a high-speed network adapter: A software perspective. Proceedings of the ACM SIGCOMM Conference, pp: 152-160.

Law and W. Kelton, 2000. Simulation Modeling and Analysis, McGraw-Hill, 3rd Edn., 2000.

Mogul, J. and K. Ramakrishnan, 1997. Eliminating receive livelock in an interrupt-driven kernel. ACM Trans. Computer Systems, 15: 217-252.

Morris, R., E. Kohler, J. Jannotti and M. Kaashoek, 2000. The click modular router. ACM Transactions on Computer Systems, 8: 263-297.

Ramakrishnan, K., 1993. Performance consideration in designing network interfaces. IEEE J. Selected Areas in Commun., 11: 203-219.

Rizzo, L., 2002. Device Polling support for Free BSD. http://info.iet.unipi.it/~luigi/polling/, online document, Feb.

Salah, K. and K. Badawi, 2003. Performance evaluation of interrupt-driven kernels in gigabit networks. IEEE Globecom 2003, San Francisco, USA, December 1-5, 2003, pp: 3953-3957.

Shivan, P., P. Wyckoff and D. Panda, 2001. EMP: Zero-copy OS-bypass NIC-driven Gigabit Ethernet Message Passing. Proceedings of SC2001, Denver, Colorado, USA, November, pp: 192-200.

Zec, M., M. Mikuc and M. Zagar, 2002. Estimating the impact of interrupt coalescing delays on steady state TCP throughput. Proceedings of the 10th SoftCOM 2002 Conference, pp: 182-188.