Journal of Computer Science 1 (4): 461-470, 2005
ISSN 1549-3636
© Science Publications, 2005

# Performance Evaluation of a QoS-Aware Mechanism for Polling-Based High-Speed Network Interfaces

Salman Al-Qahtani, Tarek Helmy and Khaled Salah
College of Computer Science and Engineering, King Fahd University of Petroleum and Mineral,
Dhahran 31261, Mail Box 413, Kingdom of Saudi Arabia

**Abstract:** The explosive growth of the high-speed multimedia networks and the widespread use of web-based related applications place new demands on the network servers. The network end systems such as PC-Router, network server and host connected to high speed links must satisfy QoS requirements for multimedia traffics such as delay and loss ratio. Most works on operating systems support for high-speed network interface have focused in increasing the throughput and decreasing the interrupt handing overhead. However, the problem is that not all traffic streams are equal in terms of QoS requirements. Multimedia applications often depend more on low-latency than on high throughput. This study first compares the performance measures of hard timer and soft timer polling schemes used for high-speed network interface with high traffic load. Then it modifies the polling-based interrupt handling for high-speed network to not just eliminate the interrupt overhead but also to guarantee the QoS requirements for multimedia traffic. Meanly we propose an input network interface mechanism, which combines the advantages of using polling interrupt handling under high traffic load and using multi-priority queues scheduling algorithm to provide the QoS requirements. 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. The performance evaluations, which are performed using a discrete event simulation, indicate that under conditions of high traffic load, the proposed system offers increased throughput and reduced latency for real time traffics.

**Key words**: High-speed networks, Multimedia traffic, NIC, Interrupts, 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

**Corresponding Author:** Tarek Helmy, College of Computer Science and Engineering, King Fahd University of Petroleum and Mineral, Dhahran 31261, Mail Box 413, Kingdom of Saudi Arabia
Tel: 966-3-860-1967, Fax: 966-3-860-2174

applications. This is actually the probability when there is no polling processing and there are no packet being processed by the protocol stack. 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, we compare the performance measures of hard timer and soft timer polling schemes used for high-speed network interface under high traffic load. Second, the polling-based interrupt handling for high-speed network are modified to not just eliminate the interrupt overhead but also to guarantee the QoS requirements for multimedia traffic. Meanly, we propose an input network interface mechanism, which combines the advantages of using polling interrupt handling under high traffic load and using multi-priority queues scheduling algorithm to guarantee some level of QoS requirements for Real Time (RT) traffic. 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. By this, we can maintain the QoS requirements by giving a higher priority to RT traffic and retain the system availability by using the polling process to handle the network interrupt. The proposed mechanisms are evaluated and compared using a discrete event simulation under high traffic load.

## A MODEL OF THE NETWORK INTERFACE

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 [1]. 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 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[1, 2].

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[2,3]. 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). This simple architecture is important because it represents the most efficient base case, but it has serious limitations. It processes packets in FCFS order and so is unable to differentiate the level of service required by each traffic types. Recognition of this limitation has prompted the architecture shown in Fig. 2, where the key idea is to filter incoming packets in multiple buffers (two in our case) based on QoS requirements.

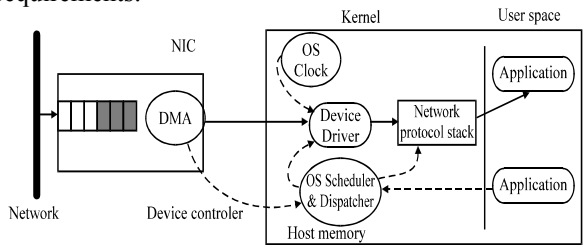


Fig. 1: The basic component of the receive part of the network interface

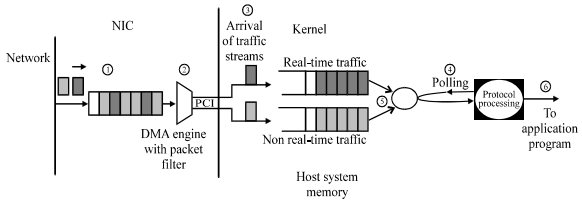


Fig. 2: Proposed QoS-aware network interface

These queues (buffers) each require access to a single resource, namely the protocol stack. When packets arrive to NIC (1), DMA, which is equipped with a preinstalled packet filtering, filters the packets (2). Then each traffic type is inserted into its queue (3). When the queues are polled (4), the appropriate scheduling scheme (5) selects the next packets to be processed by protocol stacks (6).

This proposed system model can be divided in general into two main stages:

At the first stage, the input process executes the following steps:

- Read: read a packet from input port (DMA Rx).
- Classify: classify the packet
- Enqueue: enqueue packet on appropriate queue

At the second stage, an output process performs the following steps:

- Select: select queue for next packet to transmit (processed by protocol processing).
- De-queue: de-queue the packet from this queue
- Transmit: transmit the packet to the upper application

Therefore, the scheduling problem is how to choose queue to serve in each packet transmissions with a satisfactory performance that matches its needs.

## THE POLLING PROCESS AND SCHEDULING ALGORITHM

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[4].

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[5]. 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 [5]. In fact, using hardware timers to back up soft-timers, which what we actually do in this study; 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 [4]. 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.

**Weighted round-ribbon scheduling scheme:** Polling with quota is used to poll the queues periodically. At each poll period, a finite number of packets (quota) from all queues are served. If the quota is served before the next poll arrival, the queue processing is disabled until the next poll arrival. Else, if the next poll arrives before serving the pervious quota, then a new quota

starts again. This quota is divided among the two queues such that the RT queue has $q_1$ packets per poll period while the non-Real Time (nRT) queue has $q_2$ packets per poll period, such that $0< q_1< q_{1\leq}$ Quota and $q_1+ q_{1=}$ Quota. At each poll arrival a weighted Round Robin (RR) scheduling algorithm is implemented. At the begging of each poll event the queues are served in a RR fashion where the RT queue is served first based on FCFS until it consumes its assigned quota $q_1$. Then the second queue is selected and served based on FCFS until it consumes the rest of assigned quota $q_2$, or the next poll event arrives. If the next poll event arrives, whether the second queue finishes its quota or not, the control is transferred to first queue and new quota starts and so on. Theses $q_1$and $q_2$ quotas are selected such that the RT traffic has higher weight while keeping acceptable fair for the second queue. The maximum quota per poll period is dependent on the length of the poll period and the average service time of each packet.

There could be different way to serve queues during each polling period. One way is to keep serving first queue until it finishes (queue is empty). However, due to high traffic this will cause a starvation problem for second queue packets. Another way is to keep serving first queue a finite number of consecutive polls then the second queue is served a less number of consecutive polls. However, this will result in a delay jitter due to burst serving. Therefore, the best way is to serve each queue during each period by serving the first queue with greater quota.

**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 traffic types considered are RT traffic such as interactive multimedia and nRT traffic such as file transfer. The RT traffic constraints limited to the delay.

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

Here we presents performance measurements of the implementation of NIC QoS-aware scheduling mechanisms with polling supporting using a discrete-event simulation, which is developed and implemented by C programming language.

First, the performance measures are defined and presented. Then the simulation parameters and assumption are described. After that, the event driven simulation is explained. Finally, the results of the simulation are discussed and compared.

**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 arrival 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 $=\sum$packets processed / simulation time;
- System availability = 1- ($\sum$ packets processing time +$\sum$ polling overhead time )/ simulation time;
- Delay time of $packet_i$ = departure $time_i$ - arrival $time_i$;
- System average latency of $traffic_i$ =$\sum$ $packets_i$ delay times/$\sum$ number of $packets_i$ processed;
- Blocking probability of $traffic_i$ =$\sum$ dropped $packets_i$/$\sum$ number of $packets_i$ 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 many research[4-6]. 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.

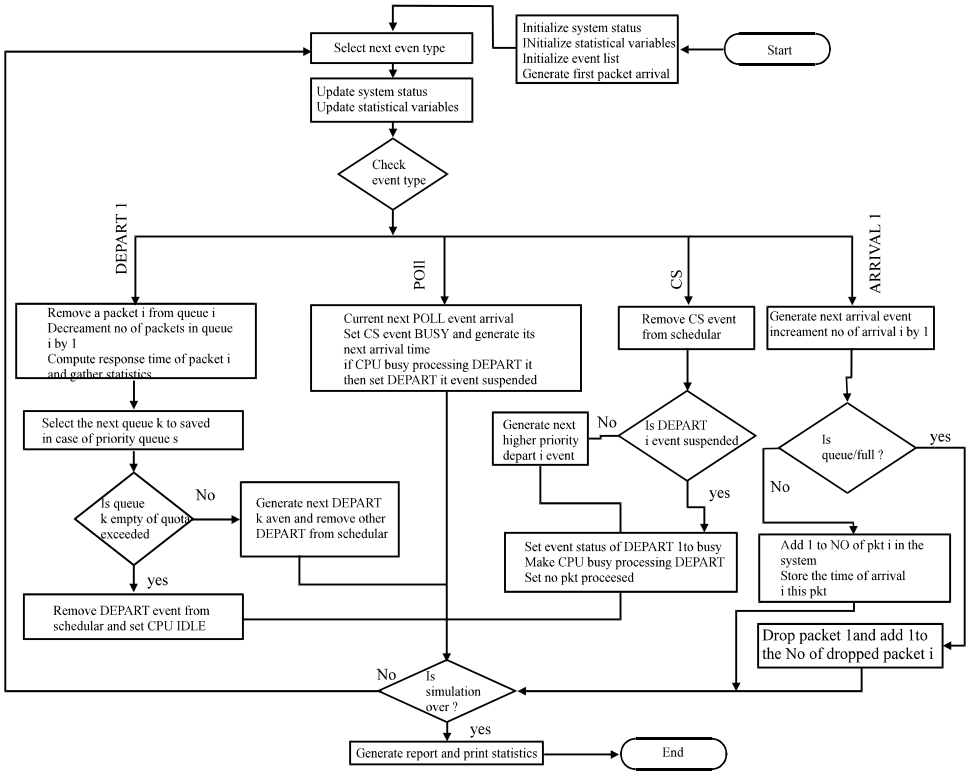


Fig. 3: Simulation flow chart

Throughout our studies, we assume the following:

- Both of service times, protocol processing or Context Switching (CS) handling, change due to various system activities[6,7]. 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 Mogul and ramkrishnan[4], 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, Morris *et al*.[6] traffic of fixed-size packets was generated back-to-back to the router. The packet's size for data can range from 64 to1500 bytes.
- The queue size of both real and nRT has a finite size of K and L packets, respectively and the common queue size is Q=K+L.
- 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.

Our simulations parameters values are based on what is reported by previous experiments[4-7]. Hard timer polling period is 20 μs while software timer has empirical distribution with minimum 2 μs and bounded by 20 μs. 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 model description:** 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[8]. There are four events for DMA Rx system. $ARRIVAL_i$, occurs when a new packet of type i arrives to host memory form the NIC. POLL, occurs when a POLL starts to check the queues for packets. CS, Indicates the completion of context switching work done by polling (checking buffers). $DEPART_i$ indicates the completion of protocol processing of one packet. Each event has a time, status and priority. An event status can be IDLE,

BUSY, or SUSPEDNDED. IDLE indicates the event has not been selected by the scheduler, i.e., not being served by the CPU. BUSY indicates the event is being served by the CPU. SUSPENDED indicates the event was BUSY but got preempted by a higher priority event. Only $DEPART_i$ event can has SUSPENDED status. The selection of the next event by the scheduler is based on the event's time, status and priority. Whenever a SUSPENDED event is selected again to run (i.e., resumed running), its finish time will incur the service times of all higher priority events which occurred between its suspension and its resumption. Figure 3 shows the simulation flow chart.

In the case of non-prioritized polling scheme, the simulation has one buffer implemented as FIFO queue. In the case of prioritized polling scheme, the simulation has two buffers each implemented as FIFO when it is given chance to be served based on a predefined scheduling scheme mentioned earlier. The simulation run ends when the total number of events reaches five millions.

**Simulation results:** This section analyzes the simulation results conducted to study the proposed system performances. Three scenarios are compared with and without packets prioritization. At each 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. The simulation results consist of two main parts. First part is to compare the performance of hard-timer polling and soft-timer polling schemes using different quota limits and the second part is to study the QoS-aware scheduling under different traffic mix situations.

**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[5] with a bound of 20 μs. The measured empirical period CDF of ST-Apache gives us the resulting distribution of soft timers delay from 2 μs until 150 μs[5]. As we state before, using harware 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[9]. Figure 4 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. 5, Fig. 6 and Fig. 7, 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. 6) and the CPU availability (Fig. 7) 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. 5). In addition, the system delay and CPU availability keep with a fixed value at higher traffic (Fig. 6 and Fig. 7). 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.

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

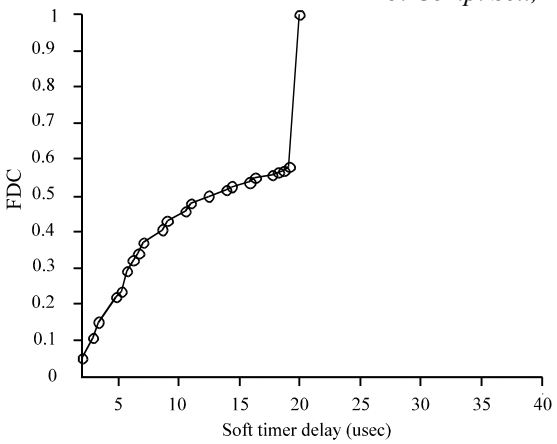


Fig. 4: Soft timer delay distribution

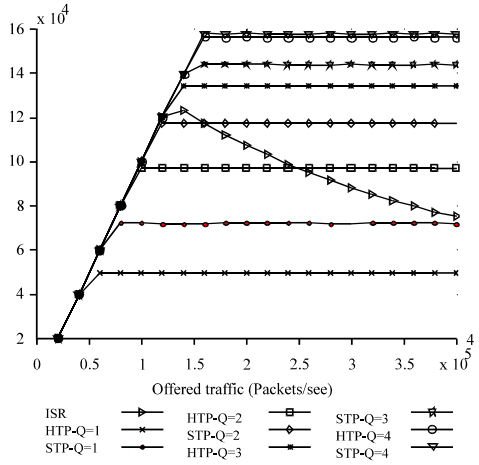


Fig. 5: System throughput

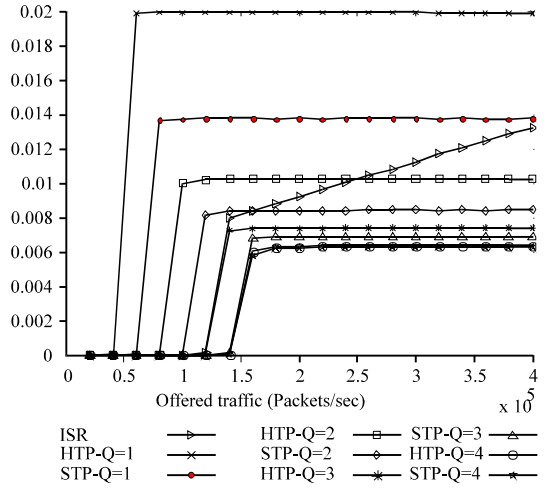


Fig. 6: System latency

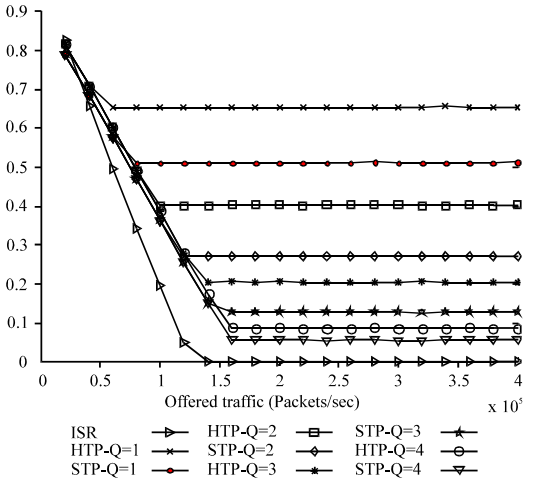


Fig. 7: CPU availability

**QoS-aware scheduling under high traffic loads:** In this subsection, the performance of the new proposed design (polling-bases NIC with traffic differentiation, denoted by P in Fig. 8-11) is studied and compared with the traditional system (polling-bases NIC without traffic differentiation, denoted by NP in Fig. 8-11) under different environments (or scenarios). For each case, we compare the system's performance in terms of system's throughput, latency, loss ratio or CPU availability. The total throughput for all cases is the same as shown in previous subsection. However, the throughput, packet delay and loss ratio of each class are dependent on the used scheduling algorithm and also depend on how the queue sizes, traffic loads of each class and quota per poll period are selected.

**Scenario 1:** RT and nRT traffic have the same traffic load and two queue sizes are used. In this scenario the total quota is 4 and the size of quota assigned to RT is 2 packets per poll period and served first while the quota assigned to nRT is $\leq 2$ packets per poll.

The queue sizes of each traffic class are (K=L=500), (K=L=1000) packets and the common queue size for all classes without differentiation are (Q=1000, Q=2000). In this experiment, as aggregated traffics (offered traffics) increase, we measure the delay using these two different queue sizes and we fix other parameters. The packet delay is most important, so it is compared and studied at these different queue sizes.

The latency of the two systems (NP and P) at different queues sizes are shown in Fig. 8. The latency of RT traffic using differentiation techniques is much lesser than its latency in case of using one common queue for both traffics classes, but this is at the expense of acceptable increase in the latency of nRT traffic which is actually delay tolerable. With respect to the queue size, as we increase queue sizes for both systems, the latency increases.

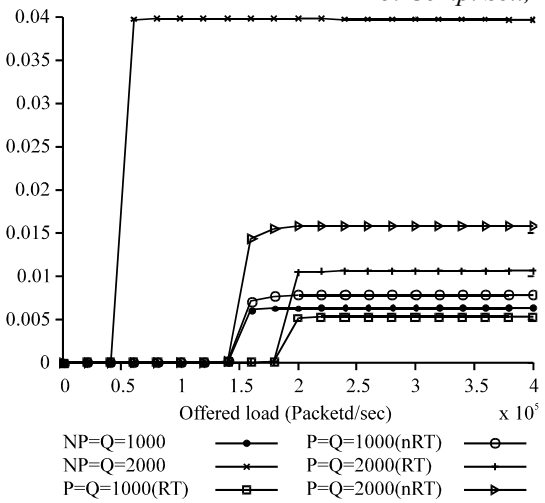


Fig. 8: System latency per traffic class

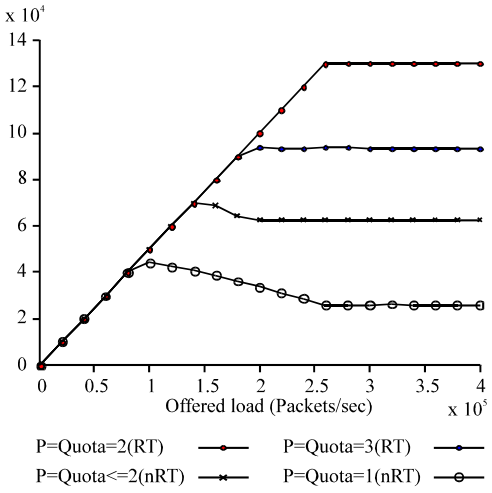


Fig. 10: Throughput per traffic class

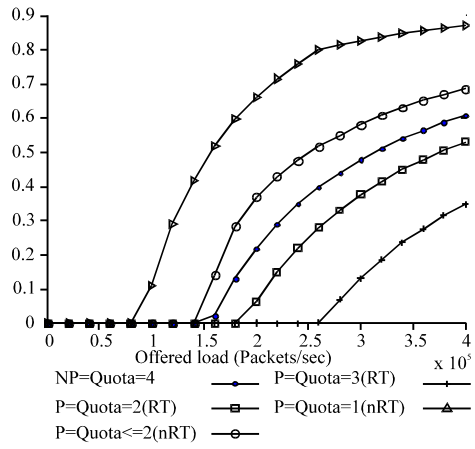


Fig. 9: Latency per traffic class

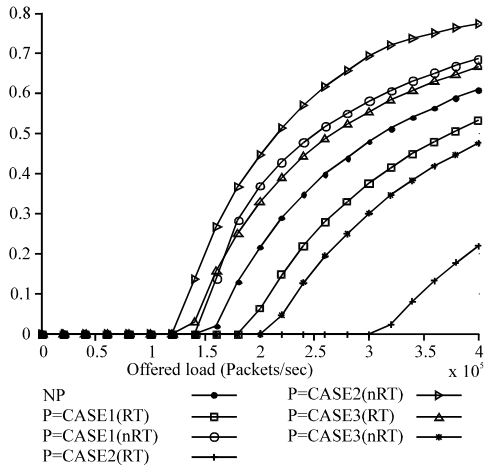


Fig. 11: Blocking probabilities

Therefore we can say that, differentiation is very important since it decreases the latency of RT traffic, but at the same time we should avoid long queue, so to avoid long delay. Choosing queue size will be based on the actual maximum delay allowed for each traffic class.

**Scenario 2:** Different Quota limits: In this experiment, the variable parameter here is the quota size of each traffic class during each poll period, other parameters are fixed. The blocking probabilities and throughput of each traffic class using different quota are compared as seen in Fig. 9 and 10. First we assume that ($q_1$=2, $q_2 \leq$2) and ($q_1$=3, $q_2$ =1). In general, serving the RT traffic first at each poll period with quota greater than nRT traffic will result in decreasing the blocking probability and increasing the throughput of RT traffic compared to the system without doing the differentiation. But, as we increase the quota assigned to RT traffic, the throughout and blocking probability of nRT traffic will become worse.

So, it is very important to choose an appropriate quota limit such that the blocking probability of RT traffic decreases while we keep the blocking probabilities of nRT traffic with acceptable limit, since we can retransmit the nRT traffic.

**Scenario 3**: Different traffic loads: In this experiment, we want to investigate the proposed system and the traditional one in terms of the blocking probabilities. We assume three cases for the aggregated traffic: case 1: RT is 50%, case 2: RT is 30% and case 3: RT is 70% of the total aggregated traffic.

As seen in Fig. 11, if the RT traffic portion is ≤ the nRT traffic portion then, the blocking probability will improve with differentiation methods, however, in case 3, the blocking probability of RT traffic will be worse than in case of traditional system without differentiation. This is because, RT traffic is limited to its queue size and nRT traffic will benefit from this, since it will not be affected by the high RT traf

fic. Therefore its blocking probability is less in this case. In case of traditional system, any traffic class with higher portion of traffic load will benefit more since it can use the whole queue (common queue) more than the traffic class with lesser traffic portion. Therefore the priority schemes will be more efficient in the case of environments with higher nRT traffic than RT traffic.

## 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[4]. 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[4]. 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[3]. Mogul and Ramakrishnan [4] 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[10]. This differs than[4], 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)[12,12]. Sometimes this scheme is known as interrupt batching in the literature[9]. 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. The authors of[13] 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[14]. 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.

All these schemes have the drawback that they do not distinguish between packets of different streams; hence applications that are latency critical suffer poor performance if the interface is being used for bulk data transfer. In addition, 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 study.

## 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 proposed QoS-aware scheduling 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 polling is used as interrupt handler. 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. This proposed system is compared with other traditional systems with only one queue for all incoming traffic.

Performance evaluations indicate that under conditions of high traffic load, the proposed system offers increased throughput, stable overload behavior and reduced latency for RT 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. Also, in the case of

multimedia traffic, we showed that the traffic differentiation will give better performance to the RT traffic which is delay sensitive at the expense of small increase in the delay of nRT traffic which is delay tolerable. 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.

## ACKNOWLEDGEMENT

We would like to thank King Fahd University of Petroleum and Minerals for supporting this research work and providing of the computing facilities. Special thanks to anonymous reviewers for their valuable comments on this paper.

## REFERENCES

1. Ramakrishnan, K., 1993. Performance consideration in designing network interfaces. IEEE J. Selected Areas in Communications, 11: 203-219.
2. 3Com Corporation, Gigabit Server Network Interface Cards 7100xx Family. http://www.costcentral.com/pdf/DS/3COMBC/DS3COMBC109285.PDF
3. Rizzo, L., 2002. Device polling support for FreeBSD. http://info.iet.unipi.it/~luigi/polling/.
4. Mogul, J. and K. Ramakrishnan, 1997. Eliminating receive livelock in an interrupt-driven kernel. ACM Trans. Computer Systems, 15: 217-252.
5. Aron and P. Druschel, 2000. Soft Timers: Efficient microsecond software time support for network processing. ACM Transactions on Computer Systems, 18: 197-228.
6. Morris, R., E. Kohler, J. Jannotti and M. Kaashoek, 2000. The click modular router. ACM Trans. Computer Syst., 8: 263-297.
7. Salah, K. and K. Badawi, 2003. Performance evaluation of interrupt-driven kernels in gigabit networks. IEEE Globecom 2003, San Francisco, USA, pp: 3953-3957.
8. Law and W. Kelton, 2000. Simulation Modeling and Analysis. McGraw-Hill, 3rd Edition.
9. BestFit, http://www.palisade-europe.com/html/bestfit.html
10. Dovrolis, C., B. Thayer and P. Ramanathan, 2001. HIP: Hybrid interrupt-polling for the network interface. ACM Operating Systems Rev., 35: 50-60.
11. 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 Conf.
12. Druschel, P., L. Larry Peterson and S. Bruce Davie, 1994. Experiences with a high-speed network adapter: A software perspective. Proc. ACM SIGCOMM Conf.
14. Alteon WebSystems Inc., Jumbo Frames, http://www.alteonwebsystems.com/products/white_papers/jumbo.htm
13. Shivan, P., P. Wyckoff and D. Panda, 2001. EMP: Zero-copy OS-bypass NIC-driven gigabit Ethernet message passing. Proc. SC2001, Denver, Colorado, USA.