Computer Communications 32 (2009) 179–188

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Implementation and experimental performance evaluation of a hybrid interrupt-handling scheme

K. Salah *, A. Qahtan

Department of Information and Computer Science, King Fahd University of Petroleum and Minerals, P.O. Box 5066, Dhahran 31261, Saudi Arabia

ARTICLE INFO

Article history:
Received 9 April 2008
Received in revised form 27 September 2008
Accepted 2 October 2008
Available online 14 October 2008

Keywords:
High-speed networks
Operating systems
NAPI
Linux
Interrupts

ABSTRACT

The performance of network hosts can be severely degraded when subjected to heavy traffic of today's Gigabit networks. This degradation occurs as a result of the interrupt overhead associated with the high rate of packet arrivals. NAPI, a packet reception mechanism integrated into the latest version of Linux networking subsystem, was designed to improve Linux performance to suit today's Gigabit traffic. NAPI is definitely a major step up from earlier reception mechanisms; however, NAPI has shortcomings and its performance can be further enhanced. A hybrid interrupt-handling scheme, which was recently proposed in Salah et al. [K. Salah, K. El-Badawi, F. Haidari, Performance Analysis and Comparison of Interrupt-Handling Schemes in Gigabit Networks, International Journal of Computer Communications, Elsevier, Amsterdam 30 (17) (2007) 3425–3441], can better improve the performance of Gigabit network hosts. The hybrid scheme switches between interrupt disabling–enabling (DE) and polling (NAPI). In this paper, we present and discuss major changes required to implement such a hybrid scheme in the latest version of Linux kernel 2.6.15. We prove experimentally that the hybrid scheme can significantly improve the performance of general-purpose network desktops or servers running network I/O-bound applications, when subjecting such network hosts to both light and heavy traffic load conditions. The performance is measured and analyzed in terms of throughput, packet loss, latency, and CPU availability.

1. Introduction

1.1. Background

Using today's general-purpose Gigabit network adapters (also termed as NICs), an incoming packet gets transferred (or DMA'd) through the PCI bus from the NIC to a circular buffer in the kernel space known as DMA Rx Ring. 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. During protocol processing, other packets may arrive and get queued into the DMA Rx Ring. Protocol processing typically involves TCP/IP processing of the incoming packet and delivering it to user applications. The packet does not need to be delivered to user applications if the receiving host is configured for IP forwarding, routing, filtering or NATing.

The performance of network hosts can be significantly degraded when subjected to heavy traffic load such as that of Gigabit networks, and thus resulting in poor host performance perceived by the user. This is because every incoming packet triggers a hardware interrupt, which involves a context switching of saving and restoring processor's state and also in a potential cache/TLB pollution. More importantly, interrupt-level handling, by definition, has an absolute priority over all other tasks. If the interrupt rate is high enough, the system will spend all of its time responding to interrupts, while nothing else would be performed. This will cause the system throughput to drop to zero. This situation is called *receive livelock* [1]. In this situation, the system is not deadlocked, but it makes no progress on any of its tasks, causing any task scheduled at a lower priority to starve or have no chance to run.

A number of schemes to mitigate interrupt overhead and resolve receive livelock exists in the literature. Among the most popular ones are normal interruption, interrupt disabling and enabling, interrupt coalescing, and polling. In normal interruption, every incoming packet causes an interrupt to trigger protocol processing by the kernel. Typically protocol processing is performed by a deferrable and reentrant high-priority kernel function (e.g., tasklet in Linux). The idea of interrupt disable-enable (a.k.a. DE) scheme [2,3] is to have the received interrupts of incoming packets turned off (or disabled) as long as there are packets to be processed 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). This means that protocol processing of packets by the kernel is processed immediately and at interrupt priority level. Any incoming packets (while the interrupts are disabled) are DMA'd quietly to protocol buffer without incurring any interrupt

* Corresponding author. Tel.: +96638604493.
E-mail addresses: salah@kfupm.edu.sa (K. Salah), kahtani@kfupm.edu.sa (A. Qahtan).

0140-3664/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2008.10.001

overhead. With the scheme of interrupt coalescing (IC) [4], 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. There are two IC types: count-based IC and time-based IC. In count-based IC, the NIC generates an interrupt when a predefined number of packets has been received. In time-based IC, the NIC waits a predefined time period before it generates an interrupt. Finally, the basic idea of polling is to disable interrupts of incoming packets altogether and thus eliminating interrupt overhead completely. In polling [1,5–8], the OS periodically polls its host system memory (i.e., protocol processing buffer or DMA Rx Ring) to find packets to process. In general, exhaustive polling is rarely implemented. Rather, polling with quota or budget is usually the case whereby only a maximum number of packets is processed in each poll in order to leave some CPU power for application processing.

In [3], we utilized both mathematical analysis and discrete-event simulation to study the performance of those most popular interrupt-handling schemes which included normal interruption, polling, interrupt disabling and enabling, and interrupt coalescing. For polling, we studied both pure (or FreeBSD-style) polling and Linux NAPI polling. The performance was studied in terms of key performance indictors which included throughput, system latency, and CPU availability (i.e., the residual fraction of CPU bandwidth left for user applications). Based on the study carried out in [3], it was concluded that no particular interrupt handling scheme gives the best performance under all load conditions. Under light and heavy traffic loads, it was shown that the scheme of disabling and enabling interrupts (DE) outperforms, in general, all other schemes in terms of throughput and latency. However, when it comes to CPU availability, polling is the most appropriate scheme to use, particularly at heavy traffic load. Based on these key observations and in order to compensate for the disadvantages of DE scheme of poor CPU availability, we proposed in [3] a hybrid scheme that combines both DE and polling. Such a hybrid scheme would be able to attain peak performance under both light and heavy traffic loads by employing DE at light load and polling at heavy load.

In [11], we presented preliminary experimental results to show that the performance of I/O bound applications can be enhanced when using the Hybrid scheme. In this paper we considerably extend the work presented in [11] and provide in-depth details of implementing the Hybrid scheme in the latest Linux version 2.6.15. The paper also shows how to measure experimentally the performance of Hybrid, DE, and NAPI. In addition, the paper addresses important implementation issues which include identifying the switching point between DE and NAPI, real-time measurement of incoming traffic rate, and the selection of proper NAPI budget or quota size during the polling period. The performance of the Hybrid scheme is evaluated in terms of throughput, latency, CPU availability, packet loss, and interrupt rate.

The rest of the paper is organized as follows. Section 2 gives a brief background and related work on the hybrid scheme. It discusses how different our hybrid scheme from those proposed in the literature. Section 3 details the inner-workings of Linux NAPI. Section 4 presents major changes required by network device driver to implement DE and Hybrid in latest version of Linux 2.6.15. Also the section addresses important issues as operation overhead, adjusting NAPI budget, identifying the cliff point, and switching mechanism. Section 5 describes experimental setup, then Section 6 presents performance measurements. Finally, Section 7 concludes the study and identifies future work.

2. Hybrid scheme

A hybrid scheme of normal interruption and polling was first proposed in [2]. Later, it was implemented and utilized in [5,8–10]. In this hybrid scheme, normal interrupt was used under both light and normal network traffic load, whereas polling was used under heavy network traffic load. In sharp contrast, our hybrid scheme (which we initially proposed in [3]) differs from previously proposed scheme in three significant ways: (1) Under light and normal loads, our hybrid scheme utilizes the scheme of DE as opposed to normal interruption which was used in [2,5,8–10]. (2) Our hybrid scheme switches between DE and NAPI based on the estimated incoming traffic rate. (3) The switching point is identified *experimentally*, rather than arbitrarily.

It was demonstrated in [3] that normal interruption performs relatively poorly under light and normal traffic loads in terms of system throughput, CPU availability, and latency. This was due to the fact that normal interruption introduces interrupt overhead for each packet arrival and thereby leaving limited CPU power for IP processing and user applications. On the other hand, DE gave acceptable performance in terms of system throughput, CPU availability, and latency under low and normal traffic loads.

To identify the severity of traffic load conditions, and as opposed to other hybrid schemes proposed in [2,5,8–10], our hybrid scheme switches between DE and NAPI based on the estimated incoming packet arrival rate. In particular, our hybrid scheme estimates the traffic rate periodically and uses two thresholds for switching. As will be discussed in Section 4.2, two thresholds are used to minimize repeated switchings (or oscillation) around the saturation point in the presence of a traffic load that is highly fluctuating at the point of saturation. Moreover, the saturation point was not identified arbitrary but using before-hand experimentation discussed in Section 4.2. On the other hand, different ways to identify severity of traffic load conditions as well as the switching point were used in [2,5,8–10].

In [2,9,10], the utilization (or level of occupancy) of application or socket receive buffers was used. A receive buffer that has a utilization of 5% indicates low traffic load, while a buffer that has a utilization of 70% indicates high utilization. Switching based on buffer utilization has major drawbacks. First, a host system has multiple buffers of interest that could potentially be used to indicate traffic overload conditions, and therefore making it impractical to implement a comprehensive solution. For example, for IP forwarding hosts, socket or protocol processing buffer can be used. In application hosts such as web servers, user application buffer is more appropriate. In particular, the protocol receiver buffer was used in [2], while in [9,10], the application receive buffer was used. Secondly, the buffer utilization can be high for a number of reasons other than traffic load, e.g. starvation or bursty traffic. Starvation of a consumer process (i.e., user application or protocol processing) is a possibility as the CPU power is being allocated for other highly important tasks or processing. In addition, instantaneous traffic burst can fill up the buffer very quickly and thus cause sharp increase in buffer utilization. Thirdly, selection of utilization levels or thresholds is somewhat and always done arbitrarily. According to [2], determining the upper and lower level of occupancy thresholds is arbitrary and in reality not a trivial task.

In [8], a watchdog timer is used for switching between normal interruption and polling. Upon a packet arrival a predefined watchdog time is started. If a packet is not removed within this predefined time through polling, an interrupt is generated. In today's network adapters or host hardware, such a feature of monitoring a packet consumption is not supported. A special hardware support has to be introduced to generate an interrupt if a packet is not removed from the DMA Rx Ring within the predefined amount of time. In addition, determining a proper value for the watchdog timer is not trivial, and often defined arbitrarily. A Small value will resort to normal interruption while a large value will resort to polling.