A Robust Device Hybrid Scheme to Improve System Performance in Gigabit Ethernet Networks

Xiaolin Chang

Dept. of Computer Science
Beijing Jiaotong University
Beijing, P.R. China
xlchang@bjtu.edu.cn

Jogesh K. Muppala

Dept. of Computer Science and Engg.
Hong Kong Univ. of Science and Tech.
Kowloon, Hong Kong
muppala@cse.ust.hk

Pengcheng Zou, Xiangkai Li

Red Flag Software Co., Ltd,
Beijing, P.R. China

Abstract—Studies of the performance of interrupt-driven operating systems in high-speed networks have brought forth the problem of receive livelock. Device hybrid interrupt-polling and interrupt coalescing are two common techniques used in general-purpose operating systems to mitigate this problem. Adaptive schemes based on local knowledge have been proposed for each technique above. However, all the schemes proposed so far are designed using heuristics. In addition, the capabilities of the proposed schemes have not been systematically compared. In this paper, we first analyze the capabilities of these schemes by investigating the relationship between key system parameters and system goodput in different packet protocol processing modes under heavy traffic load. Then we propose a robust device hybrid interrupt-polling (RHIP) scheme which achieves high system goodput, low packet loss and good latency with low consumption of CPU cycles, compared to other schemes. The key idea of RHIP is to use the recipient's buffer information to adjust the interrupt rate and the protocol processing time. We validate our analysis and design through several experiments.

Keywords— Gigabit Ethernet; Receive Livelock; polling; interrupt coalescing

I. INTRODUCTION

Most personal computers (PC) these days ship with Gigabit Ethernet interface cards [1]. The PC users expect to benefit from the higher speed supported by the interface cards and use them to connect to high-speed networks [2]. However this has not been the experience in practice. Studies on the performance of interrupt-driven operating systems (OS) on PCs connected to high-speed networks have brought forth the problem of receive livelock (RL) [3]. When this problem occurs, the CPU spends most of its cycles in handling interrupts thereby leaving significantly less CPU cycles for other tasks. The main reason for this behavior is that interrupt handling tasks have higher priority than all other tasks. This paper focuses on the RL problem in the current general-purpose OSes running on commodity off-the-shelf PCs.

Device hybrid interrupt-polling (HIP) and interrupt coalescing (IC) are two common techniques used in general-purpose OSes to mitigate the RL problem. Adaptive schemes based on local information to mitigate RL in dynamic environments have been proposed for each technique above. However, all these schemes are designed using heuristics.

Some problems in using schemes based on *ad-hoc* local-information include (1) there is no way to ensure that the system is able to achieve its maximum system *goodput*, or even know whether the system has reached its maximum *goodput*; (2) very little is known about the reasons why a scheme can or cannot significantly improve system *goodput*. Although some research studies such as in [8] have been carried out to theoretically analyze the packet reception process under each technique above, they cannot be generalized to investigate the capabilities of the existing schemes. Furthermore, many new Gigabit Ethernet NICs have implemented interrupt coalescing feature [4]. To the best of our knowledge no investigation has been carried out to compare the capabilities of the different schemes proposed to mitigate the RL problem.

In this paper we first analyze the relationship between key system parameters and the system *goodput* in different packet protocol processing modes under heavy traffic load. Then we apply the analysis results to investigate some existing adaptive schemes. Based on our investigation we propose a robust device hybrid interrupt-polling (RHIP) scheme. RHIP combines the advantages of the two techniques in order to improve system *goodput*, reduce packet loss and latency with low CPU cycle consumption over a wide range of hardware and traffic conditions. We use system *goodput* and *goodput* interchangeably in this paper. Note that ensuring fairness is beyond the scope of this paper.

Based on our analysis and experimental results we conclude that (1) in terms of improving system *goodput* under heavy traffic load, a system-state-aware HIP scheme performs well in some situations; while a system-state-aware IC scheme performs well in other situations; (2) an effective combination of these two techniques can produce the best performance. Considering that the networking subsystems are implemented differently in different OSes, we describe our analysis and design in the context of the Linux 2.6 Kernel. However, our approach can easily be extended to other OSes that employ a HIP scheme in kernel space. In addition, although most Gigabit NIC drivers have implemented HIP or IC schemes, the same scheme is implemented differently in different NIC drivers and different driver versions. In the following sections we describe our analysis and design in the context of the Linux driver for the Intel(R) PRO/1000 Family of Adapters (PCI-X), commonly called the "e1000".

The work described in this paper has been supported in part by Beijing Jiaotong University Science Foundation under 2007RC033, HK RGC under HKUST6177/04E, China MOST under 2005BA112A02 and China NORD

At the outset, we give definitions of some terms that are subsequently used in this paper. By packet loss, we mean the loss of those packets that have consumed some CPU cycles. Thus, we do not include packets that are dropped in a NIC. *Goodput* is defined as the rate at which packets are successfully delivered to and processed by the intended recipients. We compute the rate or *goodput* according to the packet size. By packet size, we mean the byte-count in the length field of the IP header. For a computer that only does packet forwarding such as a router or a firewall, the *goodput* is the rate of packets that can be sent out over another NIC attached to the computer. If the incoming data is delivered locally to applications, such as network monitoring or an application layer switch, the *goodput* is the rate at which packets are received and processed by all the applications. By interrupt, we mean hardware interrupt. By softirq, we mean software interrupt.

The rest of this paper is organized as follows. Section II presents background and related work. In Section III we present the analysis of the schemes to determine system *goodput*. Then we present RHIP in Section IV and present performance evaluation in Section V. Section VI gives the conclusions of the paper.

II. BACKGROUND AND RELATED WORK

This section focuses on the schemes that have been deployed or can easily be deployed in the current general-purpose OSes. Thus we do not consider those approaches that modify the network subsystem architecture, such as considering per flow information [5], implementing part of the networking subsystem in user space [6], or modifying the OS scheduling algorithm [7], or protocol offload with the aim to optimize the interface between the NIC and the CPU.

A. The interrupt coalescing schemes

The interrupt coalescing schemes mitigate the RL problem by directly limiting the interrupt generating rate in the NIC. The authors in [10] explored the feasibility of this approach but did not discuss how to choose the timer. The setting of the timer not only has an impact on the system *goodput* but also on the latency. The authors in [11] proposed adjusting the interrupt rate based on the socket buffer utilization. However, this scheme suffers from a problem in the uniprocessor systems. The providing and consuming processes of the socket buffer packets are asynchronous. Thus the buffer utilization may be zero at the end of the consuming process. It is possible that this information is sampled to adjust the interrupt rate each time. Actually, the buffer utilization is high at the end of the providing process. Then the following adjustment is wrong, leading to the *goodput* fluctuation.

B. The existing device hybrid schemes

The HIP approach is a pure software approach in which an interrupt mechanism is used under normal network traffic load and a polling mechanism is used under heavy network traffic load in order to indirectly limit the interrupt rate. A HIP scheme needs to address two issues under heavy traffic load. First, it must strike a good balance between the time in the interrupt mode and the time in the polling mode. Secondly, it

must strike a good balance between the protocol processing time and the application processing time. Failure to strike a proper balance in either case may result in packets being dropped in *rcv_buffer*. The authors in [12] observed the effects of polling time on the system performance. They proposed to adjust the polling time directly based on the observed packet inter-arrival rates. The problem with this adjustment is that it ignores the second issue above, possibly resulting in packets being dropped in *rcv_buffer*.

The authors in [3] tried to strike the proper balance as mentioned above by limiting the maximum number of packets that can be processed in a protocol processing round. Linux New API (NAPI) [9] is an implementation of this key idea. Other OSes such as FreeBSD, OpenBSD and Microsoft Windows have also used this key idea. In the remainder of this section we first describe NAPI. Then we discuss some enhancements that have been proposed in the literature.

Figure 1 Packet Reception Process

C. NAPI

To simplify the description, we focus on the UDP/IP protocol suite. From the software point of view, Figure 1 disregarding the shaded parts describes the path traversed by an incoming packet from the DMA-capable NIC to the end recipients. The steps are as follows:

- (1) Incoming packets are first put into the DMA ring by the NIC. The CPU is not involved in this action. After putting a packet in the DMA ring, the NIC raises an RX interrupt. The NIC drops the incoming packets when it cannot find free space in the DMA ring.
- (2) When the CPU receives the interrupt signal, it invokes the RX interrupt routine. This routine, which is part of the NIC driver, only disables the RX interrupt and posts a softirq. The RX routine is executed at interrupt priority. The softirq is invoked immediately at the end of the interrupt routine.
- (3) Packet protocol processing. The RX softirq routine is responsible for protocol processing, which is non-preemptible, but can be interrupted. It picks up packets from the DMA ring and puts them into *rcv_buffer*. R_m denotes the rate at which the packets are transferred. Its

value is determined by the traffic load, the PCI bus/the dedicated bus, the CPU cycles required for protocol processing a packet and the available CPU cycles.

Budget (B_p) denotes the maximum number of packets that are allowed to be pulled out of the DMA ring in an RX softirq invocation. Its value is fixed in NAPI. At the end of the RX softirq execution, the RX interrupt is re-enabled if there is no new packet arrival in the DMA ring. If an RX softirq invocation cannot empty the DMA ring, then a kernel thread, *ksoftirqd*, is activated to handle the protocol processing of the leftover packets. *ksoftirqd* works by invoking the RX softirq routine. Although preemption is disabled during the RX softirq routine execution, *ksoftirqd* can be preempted before and after the routine execution. In default, the priority of *ksoftirqd* is less than the priority of normal user-space applications. Such setting makes it possible for applications to preempt *ksoftirqd* before the basic quantum duration of *ksoftirqd* is exhausted.

rcv_buffer has different meanings in different contexts. If the computer is for packet forwarding, *rcv_buffer* denotes the incoming buffer of another NIC, whose driver picks up and processes the packets. If the computer delivers packets locally such as packet capture, *rcv_buffer* denotes the socket receiving buffer (or a predefined buffer). User-space applications pick up and process the packets from *rcv_buffer*. We study the second case (local delivery to the user-space applications) in the rest of the paper.

D. Enhancement

A static B_p may either result in the balance failure mentioned above or result in a large average CPU cycle consumption for taking a packet from the DMA ring to the application. Thus, a HIP scheme must consider the system state in order to improve system performance. The system state is determined by a number of factors, such as the dynamics of the incoming packets (packet size, packet inter-arrival times, burst size), the characteristics of the OS and the hardware resources available within the computer. The system state is difficult to be measured. *Q_{off}^{on} Polling* [3] and *QAPolling* [13] both use the socket buffer information, which changes with the system state, to improve system *goodput*. *Q_{off}^{on} Polling* uses the current socket buffer information to decide whether to disable the packet protocol processing or not. The nature of the “disable” operation is setting B_p to zero. The on-off operations may lead to *goodput* fluctuation.

Observing the drawback of *Q_{off}^{on} Polling*, *QAPolling* uses the buffer information to adaptively adjust B_p in order to improve system performance over a wide range of hardware and traffic conditions. It decreases B_p when the current *rcv_buffer* utilization (*bufUtil*) is larger than a threshold (*buf_{upper}*) and increases it when the maximum *rcv_buffer* utilization in the sampling interval is below a threshold (*buf_{low}*). When the maximum *rcv_buffer* utilization in the sampling interval is between these two thresholds, B_p is micro-adjusted according to the maximum *rcv_buffer* utilization in two consecutive sampling intervals. The shaded upper part in Figure 1 is *QAPolling* scheme. It is possible that all connections are closed with a small B_p , impacting the system response to the packets

arriving later or leading to the possible packet dropping in NIC when suddenly a large number of packets arrive. The daemon program aims to eliminate these problems.

III. GOODPUT AND PACKET LOSS ANALYSIS

In the following analysis, we assume that (1) there is only one receiving application with normal priority in the system; (2) packets arrive at a constant rate; (3) both the DMA ring size (B_{DMA}) and *rcv_buffer* size are larger than B_p ; (4) the PCI bus is not a bottleneck; (5) there is only one CPU.

A. When NAPI is enabled

Some variables which are used in the subsequent analysis are first defined below:

Notation	Description
T	the time interval between two consecutive interrupts
T_p	the CPU time for protocol processing a packet
T_{refill}	the CPU time for L2 cache refilling
T_a	the CPU time for user-space application processing a packet, including the cost of system call; thus when cache miss occur, T_a also includes T_{refill}
T_{irr}	the CPU time for an ISR execution
T_c^a T_c^k	the CPU time for context switching the user-space application, kernel thread <i>ksoftirqd</i> , respectively
T_c^i	Interrupt overhead, including hardware overhead and software overhead
B_p	the maximum number of packets pulled out of the DMA ring in an RX softirq invocation
B_a	the maximum number of packets processed by the application in a scheduling round, equal to (the application quantum duration)/ T_a
λ	the packet arrival rate at the DMA ring, thus when PCI bus is not bottleneck, λ is the packet arrival rate at NIC
k_p	the number of packets picked up from the DMA ring in an RX softirq invocation, not more than B_p
k_a	the number of packets that the application can process in a scheduling round, not more than $\min\{B_a, k_p\}$
γ	the number of packets in the DMA ring at the beginning of the protocol processing
G	the system <i>goodput</i>

Definition. The system achieves its maximum *goodput* when its *goodput* cannot be improved further by adjusting system parameters, such as B_p and T .

Proposition 1. Assume that $T_p < 1/\lambda$. Then the RX interrupt is re-enabled after one softirq invocation under constant heavy

traffic load when $B_p > \frac{\gamma}{1 - T_p \lambda}$.

Proof. $T_p < 1/\lambda$ indicates that there is no new packet arrival in the DMA ring when the protocol processing of $\gamma + \frac{T_p \gamma}{\frac{1}{\lambda} - T_p}$ packets are completed. The second term in the above

expression is the number of packets that arrive while the softirq routine is processing the packets in the DMA ring buffer. According to the description in Section II, the RX interrupt is re-enabled at this point. As long as the *budget* B_p is greater than the above value, $\frac{\gamma}{1-T_p\lambda}$, the RX interrupt will be re-enabled after one RX softirq invocation. ■

$$\text{Thus, } k_p = \min \left\{ B_p, \frac{\gamma}{1-T_p\lambda} \right\} \quad (1)$$

Proposition 2. If $T_p \geq 1/\lambda$ then the RX interrupt is never enabled under constant heavy traffic load.

Proof. $T_p \geq 1/\lambda$ indicates that there is at least one new packet arrival in the DMA ring during the protocol processing of packets in the ring by the softirq routine. Thus the DMA ring never gets emptied, and hence the RX interrupt never gets enabled before the *budget* runs out. ■

Thus, the protocol processing can be divided into three modes of operation as follows:

- (1) $T_p \geq 1/\lambda$
- (2) $T_p < 1/\lambda$ and $B_p \leq \frac{\gamma}{1-T_p\lambda}$
- (3) $T_p < 1/\lambda$ and $B_p > \frac{\gamma}{1-T_p\lambda}$

Now we analyze system *goodput* in the above three modes.

1) Goodput in the first two modes

The packet processing time line in the first two modes is depicted in Figure 2. The problem of maximizing the system *goodput* in the first two modes can be formally specified as:

$$\begin{aligned} \text{maximize } G &= \frac{k_a}{T_c^a + T_c^k + B_p T_p + k_a T_a} \\ \text{subject to } k_a &\leq k_p = B_p \end{aligned} \quad (2)$$

Figure 2 Packet processing time line when *ksoftirqd* is active

Remark 1. *rcv_buffer* may overflow in the second mode. Although *ksoftirqd* is preemptible, the process of protocol processing k_p packets is non-preemptible. That is, even if the time quantum allocated to *ksoftirqd* is used up, *ksoftirqd* still occupies the CPU until the protocol processing of k_p packets is completed. However, no matter whether there is packet in *rcv_buffer*, the application must be put in the CPU waiting queue if its time quantum is used up. $B_p < \frac{\gamma}{1-T_p\lambda}$ indicates that the RX interrupt is not re-enabled after the protocol processing

of k_p packets is completed. Thus, if $B_a < \frac{\gamma}{1-T_p\lambda}$, the application can not empty *rcv_buffer* in a scheduling round when B_p is increased to be larger than B_a . Then *rcv_buffer* will overflow.

Proposition 3. Assume that *rcv_buffer* is very large. If *rcv_buffer* overflows, then the system *goodput* can be improved.

Proof. Buffer overflow means that some CPU cycles consumed by packet protocol processing are wasted. By reducing B_p , some of these wasted CPU cycles can be saved for packet application processing. That is, decreasing B_p results in increase of k_a . ■

Proposition 4. Assume that an application can process at least one packet in an application scheduling. If the system is in the second mode, then there exists at least one B_p that can avoid *rcv_buffer* overflow.

Proof. The above discussion has mentioned that *ksoftirqd* cannot preempt the application. Thus, as long as the application processing rate is not less than the protocol processing rate, the *rcv_buffer* will not overflow. That is, as long as $k_a \leq B_p$, the *rcv_buffer* will not overflow. Since $1 \leq k_a$, there is no *rcv_buffer* overflow when $B_p = 1$. ■

When *rcv_buffer* does not overflow, we get

$$G \leq \frac{1}{\frac{T_c^a + T_c^k}{B_p} + T_p + T_a} \quad (3)$$

Remark 2. G may not increase with increasing B_p for a specific environment, determined by hardware and software configuration and traffic characteristics (packet size, packet inter-arrival times, burst size). In a specific environment, T_c^a and T_c^k are fixed. During the protocol processing, any packet in DMA ring is a new packet for CPU. Thus, T_p can be regarded as fixed. But the application processing is different. Any packet in *rcv_buffer* has ever been visited by CPU. That is, part of the packet information has ever been put into L2 cache. When B_p is larger, L2 cache missing rate is high. We use Oprofile tool to trace the system and find that when B_p is larger than a value, L2 cache missing rate of the kernel function *skb_copy_datagram_iovec* is high. Thus, we guess T_a is not fixed. There may be other reasons for G decrease, such as scheduling. We leave it for future work.

2) Goodput in the third mode

The packet processing time line in the third mode is depicted in Figure 3 and Figure 4.

The problem of maximizing the system *goodput* in the third mode can be formally specified as:

$$\text{maximize } G = \frac{1}{\frac{T_c^a + T_c^k + T_c^i + T_x + T_{itr}}{k_a} + \frac{k_a T_p}{k_p} + T_a} \quad (4)$$

$$\text{subject to } k_a \leq k_p = \frac{\gamma}{1 - T_p \lambda}$$

Proposition 5. Assume that the PCI bus is not a bottleneck and $B_{DMA} > B_p$ and the RX interrupt is re-enabled after each RX softirq invocation. Then if an arriving packet is dropped, it must be dropped in *rcv_buffer* instead of in the NIC.

Note that during the protocol processing of k_p packets the DMA memory occupied by these k_p packets cannot be re-used by the NIC until the processing is completed.

Proof. We set $T_p = T_p k_p$, $T_{NP} = T_x + k_a T_a$, $T = T_p + T_{NP}$.

(1) We first prove that there is no packet dropped by the NIC in T_p . Assume that a packet is dropped. Then there is at

least one new packet arrival in the DMA ring after $\frac{\gamma}{1 - T_p \lambda}$ ($\leq B_p$) packets are removed from the DMA ring. According to the description in Section II, the Rx interrupt is not enabled. This is a contradiction.

(2) We prove that there is no packet dropped by the NIC in T_{NP} . Assume that a packet is dropped. Then there are B_{DMA} packets in the DMA ring at the beginning of the next T . However, at most B_p packets can be processed in a protocol processing. Thus, the RX interrupt is not enabled at the end of the RX softirq routine. This is a contradiction. ■

Figure 3 Packet processing time line when $k_a = k_p = \frac{\gamma}{1 - T_p \lambda}$

Figure 4 Packet processing time line when $0 < k_a < k_p = \frac{\gamma}{1 - T_p \lambda}$

Proposition 6. Assume that the PCI bus is not a bottleneck and $B_{DMA} > B_p$. Then if there is no overflow in *rcv_buffer* and the RX interrupt is re-enabled after one RX softirq invocation (that is, the packet reception process is as in Figure 3), then the system gets its maximum *goodput*.

Proof. An arriving packet is dropped if and only if DMA ring overflows or *rcv_buffer* overflows. From Proposition 5, we know that there is no packet dropped in NIC, that is, the DMA ring does not overflow. Thus, if there is no overflow in *rcv_buffer*, the application processes all arriving packets. That is, the system achieves its maximum *goodput*. ■

Proposition 7. Assume that the packet processing time line is as in Figure 4 and G is to be improved by reducing B_p . Then,

the system must be in the second mode or oscillate in between the second and third modes if there is improvement in G .

Proof. Because the packet processing time line is as shown in Figure 4, then $\frac{\gamma}{1 - T_p \lambda} = k_p \geq k_a$. Thus in order to reduce k_p , B_p must be reduced to be less than $\frac{\gamma}{1 - T_p \lambda}$. ■

Note that if decreasing B_p cannot make the system in the second mode, decreasing B_p may not prevent *rcv_buffer* overflow.

Proposition 8. Increasing T may not prevent *rcv_buffer* overflow in Figure 4.

Proof. $T = T_x + k_p T_p + k_a T_a$. Then $\gamma = (T_x + k_a T_a) \lambda$ and $\frac{(k_a T_a + T_x) \lambda}{1 - T_p \lambda} = k_p \geq k_a$. Then $T_a + \frac{T_x}{k_a} \geq \frac{1}{\lambda} - T_p$ (5)

Thus when the packet arrival rate λ is large or T_a is large, it is possible that $(T_a + \frac{T_x}{k_a})$ is always larger than $(\frac{1}{\lambda} - T_p)$. That is, $k_p = k_a$ is impossible. In this situation increasing T cannot remove *rcv_buffer* overflow. ■

Remark 3. Assume that the system is as in Figure 4. Then increasing T requires large *rcv_buffer* unlike decreasing B_p .

B. When NAPI is disabled

Figure 5 Packet processing time line when $0 \leq k_a \leq k_{p2} \leq k_{p1} \leq B_p$

The interrupt is not disabled during the interrupt routine execution. In an interrupt routine execution, k_{p1} ($\leq B_p$) packets are removed from the DMA ring and put into a temporary queue, which is in networking system of Figure 1. If no interrupt signal arrives when all the packets in the DMA ring are removed, the protocol processing begins by taking k_{p2} ($\leq k_{p1}$) packets from the temporary queue and putting them into *rcv_buffer*. If still the interrupt signal arrival does not arrive, then application processing begins. The packet processing time line is depicted in Figure 5. The packet reception process is given similar to that in the third mode. The difference is that in the third mode the protocol processing of $\frac{\gamma}{1 - T_p \lambda}$ packets

cannot be interrupted; but when NAPI is disabled, the protocol processing can be interrupted. Thus less CPU time is left for packet protocol processing and application processing under heavy traffic load. Proposition 8 can be applied to this situation.

IV. RHIP SCHEME

The above analysis validates the design of *QAPolling*, an adaptive pure HIP scheme, in terms of improving *goodput*.

However, the analysis also shows its inefficiency. In the current Linux kernel, $T_p \geq 1/\lambda$ seldom occurs and the default B_p is larger than $\frac{\gamma}{1-T_p\lambda}$ under heavy traffic load. Thus the system is in the third mode. $\frac{\gamma}{1-T_p\lambda}$ is an increasing function

of λ . When λ is large, the set of $[1, \frac{\gamma}{1-T_p\lambda}]$ is large and then *QAPolling* algorithm can make B_p varying in this set. When λ is small, the set of $[1, \frac{\gamma}{1-T_p\lambda}]$ is small and then *QAPolling*

algorithm cannot make B_p varying in the set. Then the system oscillates between in the second mode and in the third mode. In this situation, it is possible that the average CPU cycles for processing a packet under *QAPolling* is larger than under an IC scheme.

The analysis in Section III also indicates the inefficiency of increasing T when λ is large. Thus, we propose Robust HIP (RHIP) scheme, represented by the shaded parts in Figure 1. Figure 6 describes the RHIP algorithm, which is an enhancement to *QAPolling*, combining the advantages of HIP and IC techniques to adjust the CPU cycle allocation. The deciding conditions are same as in *QAPolling*. The differences between *QAPolling* and RHIP include (1) B_p is not decreased until T is increased to a pre-defined value (set to 1/8000second in our experiments); (2) T is decreased only when B_p has been increased to a predefined value and there is no *rcv_buffer* overflow; (3) in the micro-adjustment period, as long as *ksoftirqd* is inactive, we increase B_p . The first two differences are based on Proposition 8, Remark 3 and the discussions in this section. The third difference is based on Proposition 6.

```

Before a packet is put into socket buffer
1. if (bufUtil > buf_upper) then
2.     if (1/T > 8000) then
3.         Increasing T
4.     else
5.         Decreasing B_p

Per each interval
1. If ksoftirqd is inactive in last interval then
2.     goto 6
3. else if ( bufUtil is never above buf_low in last interval ) then
4.     goto 6
5.     else if ( bufUtil is never above buf_upper in last interval ) then
6.         if (B_p < 300) then
7.             Increasing B_p
8.         else
9.             Decreasing T

```

Figure 6 RHIP Algorithm

The easy deployment of *QAPolling* has been discussed in [13]. Compared to *QAPolling*, the additional work for deploying RHIP is to make some modifications to the NIC driver.

V. PERFORMANCE EVALUATION

In this section, we carry out experiments to validate the analysis presented in Section III and the effectiveness of RHIP scheme in the Gigabit Ethernet networks with 1 Gbps. We start with a description of the experimental setup and then proceed to present the results.

Figure 7 Experimental setup

Our experimental platform, shown in Figure 7, consists of two end systems (C1, C2). These two computers are connected by a Gigabit Ethernet switch. The hardware configurations are given in Table I. The PCI bus is not a bottleneck. Unless otherwise specified, Hyper-Threading (HT) is disabled in C1. We evaluate each scheme in C1. C2 is used as the packet-generator, sending out as many packets as possible such that the full load to C1 can be sustained. There is only one program in C1 to receive the packets from C2. They all run Asianux 2.0 [14], whose kernel is upgraded to 2.6.18. All the traffic is UDP/IP based. The main reason to select the UDP protocol instead of TCP is that the flow control and congestion avoidance algorithms defined in TCP protocol may restrict the packet generating rate. To emulate the packet application processing such as storing, the application in C1 performs 200 floating-point multiplications before dropping the received packet.

TABLE I. COMPUTER CONFIGURATIONS

Hardware	C1	C2
BIOS version	SN94510J.86A.0016.2005.0329.1458	NT94510J.86A.3191.2005.1112.1343
CPU	PD 820/2.8GHz	P4 531/3.0GHz
Connection to chipsets	dedicated CSA bus, theoretical bandwidth of 266 Mbytes/sec	dedicated CSA bus, theoretical bandwidth of 266 Mbytes/sec
RAM	1GB	512MB
Gigabit NIC	Intel 82571	Intel 82571
Driver version	E1000 v7.1.9	e1000 v7.1.9

T is varied in [1/8000, 1/2000] second in default in the e1000 driver. This setting reduces the system response under light/medium traffic load. The ping rate can be improved by 200% when an interrupt is generated for each packet arrival. The experiment results in [13] are got with $T=1/8000$ s. In this paper, we do the experiments by allowing an interrupt per packet arrival when there is no RL problem. In one softirq invocation in Linux kernel 2.6, each *softirq* routine is executed *MAX_SOFTIRQ_RESTART* (set to 10 in default) times and at most *netdev_budget* (set to 300 in default) packets are protocol processed in a routine execution. That is, an RX softirq invocation can pick up at most 3000 ($B_p = \text{MAX_SOFTIRQ_RESTART} \times \text{netdev_budget}$) packets from the DMA ring in default. Thus, the upper bound of B_p is set to 3000. In all the experiments and schemes, we vary B_p by varying *netdev_budget*. To avoid the effect of *netdev_max_backlog* in Linux kernel 2.6.18 on the experiment results, *netdev_max_backlog* is set to 3000. In addition, the

Figure 8 Goodput versus B_p in C1

kernel stops the protocol processing in default when the protocol processing time is beyond 1ms. We remove this limitation.

Unless otherwise specified, the values of other parameters used by *QAPolling* and RHIP are set as follows: $buf_{low}=3\%$, $buf_{upper}=50\%$, $rcv_buffer=8000000$ bytes, $T=100$ ms, $\alpha=2$, $\beta=1$, # of DMA ring count = 4096. For a detailed specification of the parameters for *QAPolling*, readers are referred to [13]. All the parameter settings in *QAPolling* are used in RHIP. In addition, RHIP decreases T by 10% and increases T by 500.

A. Effect of B_p on G

We do experiments by setting `MAX_SOFTIRQ_RESTART` to 1 and varying `netdev_budget` from 1 to 3000. We use four different packet sizes, viz. 64, 128, 150 and 256 bytes. Figure 8 shows the *goodput* versus B_p for different values of B_p .

Now we give detailed explanations of the experiment results for the packet size of 150 bytes. Assuming the initial value of B_p is 3000. When B_p is larger than 900, we observe that the system is in the third mode and there is no *goodput* change with the decreasing B_p . *Goodput* increases with the decreasing B_p when B_p is varied in the interval [70,900]. When B_p is varied in the interval [750, 900], the system oscillates between in the third mode and in the second mode. This oscillation is observed by using “*mpstat*” to observe the interrupt rate. When B_p is varied in the interval [600, 750], the system is in the second mode but *rcv_buffer* overflows. When B_p is varied in the interval [70, 600], the system is in the second mode and there is no *rcv_buffer* overflow. When B_p is less than 70, we could see the *goodput* decreases with the decreasing B_p . These results confirm Proposition 4, Remark 1 and Proposition 7. Similar explanations can be offered for the experimental results for other packet sizes. In all the experiments there is no *rcv_buffer* overflow when B_p is less than 500.

B. Evaluating each scheme under various packet size

The experiments in this subsection evaluate four schemes: NAPI, Adaptive IC (AIC) scheme, *QAPolling*, and RHIP. The main idea of AIC is to apply the adjusting algorithm in *QAPolling* to adjust the interrupt rate instead of B_p . In AIC, the low bound of the interrupt rate is 1 each second.

In order to show the ability of each scheme with less modification to kernel, we perform the experiments by setting `MAX_SOFTIRQ_RESTART` to 10, the default setting. Then `netdev_budget` is varied in the interval [1,300]. All the packets in an experiment have the same size. We test seven packet

sizes, viz. 46, 64, 128, 512, 1024, 1200, and 1500 bytes. Figure 9 shows the results. “SRate” denotes the sending rate of C2. Table II gives the percentage of the CPU idle time under different schemes when packet size is no less than 512 bytes. We can observe that:

Figure 9 Goodput in C1 versus packet size

Table II Percentage of the time when CPU is idle

Packet size (byte)		512	1024	1200	1500
Scheme					
NAPI		0%	0%	0%	0%
QAPolling		0%	0%	0%	0%
RHIP without adjusting		16%	43%	48%	52%
RHIP		16%	48%	52%	57%

- Under the pure NAPI scheme, the default `netdev_budget` setting leads to the system operating in the third mode and then the RX interrupt is re-enabled at the end of protocol processing. When packet size is less than 1500 bytes, so many interrupts per second results in less CPU cycles for application. Thus, the system *goodput* is very small. In all experiments under NAPI *rcv_buffer* is overflow. This indicates the inefficiency of a static HIP scheme.
- *QAPolling*’s performance is comparable to RHIP in terms of improving system *goodput*, but at the cost of consuming more CPU cycles. Table II shows the significant reduction in CPU cycle consumption under RHIP. When packet size is less than 512 bytes, RHIP behaves like *QAPolling*.
- AIC performs just as well as RHIP in terms of improving the system *goodput* with low CPU consumption when packet size is not less than 512 bytes. When packet size is less than 512 bytes, AIC cannot avoid *rcv_buffer* overflow and does not perform well. This confirms Proposition 8.

C. Various application workload

The experiments in this subsection demonstrate the superiority of an interrupt coalescing scheme over a pure HIP scheme in some situations. Just as before, MAX_SOFTIRQ_RESTART is to 10. The packet size is set to 1500 bytes. We perform experiments with the four schemes, respectively, by increasing the application workload. We vary the times of floating-point multiplication from 0 to 2000 to emulate the increasing application workload. Figure 10 shows the *goodput* under each scheme versus application workload. RHIP and AIC perform best. The low CPU cycle consumption in protocol processing saves much CPU time and then gives more chance for application processing. *QAPolling* performs better than NAPI by decreasing BP to free some CPU cycles for application processing.

Figure 10 Goodput in C1 versus application workload

D. Ping Latency

The experiments in this subsection evaluate RHIP and *QAPolling* in terms of latency. We measure Round Trip Time (RTT) of ICMP packets using ping. C2 sends 1500-byte packets as many as possible to C1. We ping C1 from another computer with 100Mbps Ethernet Card. Figure 11 shows the ping latency variation over time when *QAPolling* and RHIP are employed, respectively. The latency performance of AIC is same as that of RHIP. The significant variation of latency under *QAPolling* is caused by the greater CPU cycles required for packet protocol processing.

Figure 11 Ping Latency in C1

VI. CONCLUSIONS

In this paper we evaluate the existing schemes for mitigating RL problem by analyzing the relationship between the key system parameters and the system *goodput* under heavy

traffic load in different packet protocol processing modes. Observing the advantages and drawbacks of these schemes in mitigating RL problem, we propose a new scheme, RHIP. The key idea is to adaptively adjust the interrupt rate and the protocol processing time according to the system state. The experiment results support the analysis and demonstrate the superiority of RHIP over a wide range of hardware and traffic conditions.

Note that all the discussions in this paper are in the context of using PCI-X NIC, that is, PCI bus is not bottleneck. When PCI bus is bottleneck, all the discussions can be applied except that λ is the packet arriving rate at the DMA ring.

Although the work in this paper aims to improve the system performance automatically, it also gives more guidelines for manual adjustment, compared to *QAPolling*. In addition, although this paper focuses on RL problem in kernel, the method can be applied to RL problem in user space applications.

REFERENCES

- [1] N. L. Binkert, L. R. Hsu, A. G. Saidi, R. G. Dreslinski, A. L. Schultz, and S. K. Reinhardt, "Performance Analysis of System Overheads in TCP/IP Workloads," In Proc. 14th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT), Sept. 2005.
- [2] Luca Deri, "Improving passive packet capture: Beyond device polling," In Proceedings of the Fourth International System Administration and Network Engineering Conference (SANE 04), Sept. 2004.
- [3] J. C. Mogul, and K. K. Ramakrishnan, "Eliminating receive livelock in an interrupt-driven kernel," In Journal of ACM Transactions on Computer Systems, vol. 15, no. 3, pp. 217–252, Aug. 1997.
- [4] W.F. Wang, J.Y. Wang, and J.J. Li, "Study on Enhanced Strategies for TCP/IP Offload Engines," In Proc. 11th IEEE ICPADS, 2005.
- [5] P.Druschel, and B.Gaurav, "Lazy Receive Processing (LRP), A Network Subsystem Architecture for Server Systems," In Proc. 2nd USENIX Symp. On Operating Systems Design and Implementation, Oct. 1996.
- [6] B. Leslie, P. Chubb, N. Fitzroy-Dale, S. Gotz, C. Gray, L. Macpherson, D. Potts, Y. Shen, K. Elphinstone, and G. Heiser, "User-level device drivers: Achieved performance," In J. Comput. Sci. & Technol., vol. 20, Sept. 2005.
- [7] Y.T. Zhang, and R. West, "Process-Aware Interrupt Scheduling and Accounting," In Proc. 27th IEEE RTSS, Dec. 2006.
- [8] K. Salah, and K. El-Badawi, "On Modelling and Analysis of Receive Livelock and CPU Utilization in High-speed networks," In International Journal of Computers and Applications, 2006.
- [9] J.H. Salim, R. Olsson, and A. Kuznetsov, "Beyond Softnet," In Proc. Linux 2.5 Kernel Developers Summit, San Jose, CA, USA, Mar. 2001.
- [10] I. Kim, J. Moon, and H.Y. Yeom, "Timer-Based Interrupt Mitigation for High Performance Packet Processing," In Proc. 5th International Conference on High-Performance Computing in the Asia-Pacific Region, 2001.
- [11] A. Indiresan, A. Mehra, and K.G. Shin, "Receive livelock elimination via intelligent interface backoff," TCL Technical Report, University of Michigan, 1998.
- [12] D. Constantinou, T. Brad, and R. Parmeswaran, "HIP: Hybrid interrupt-polling for the network interface," In Proc. ACM SIGOPS Operating Systems Review, 35(4):50–60, Oct. 2001.
- [13] X.L. Chang, J.K. Muppala, "A Queue-based Adaptive Polling Scheme to Improve System Performance in Gigabit Ethernet Networks", In Proc. 26th IEEE IPCCC, Apr. 2007.
- [14] <http://www.asianux.com>.