Analysis of Interrupt Coalescing Schemes for Receive-Livelock Problem in Gigabit Ethernet Network Hosts

Xiaolin Chang¹, Jogesh K. Muppala², Zhen Han¹, Jiqiang Liu¹

¹Dept. of Computer Science Beijing, JiaoTong University, Beijing, P.R. China

²Dept. of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong
{xlchang,zhan,jqliu}@bjtu.edu.cn, muppala@cse.ust.hk

Abstract—Interrupt coalescing (IC) technique has been used in general-purpose operating systems to mitigate Receive Livelock (RL) problem in Gigabit Ethernet Network hosts. Schemes for dynamically tuning the interrupt coalescing behavior of a communication interface based on traffic load or system state have been proposed. However, all the existing IC schemes are designed using heuristics. In this paper we present an analytical model for the IC technique and carry out a detailed study of existing IC schemes in terms of their performance characteristics including system goodput, CPU consumption and latency. We validate our analysis through measurement-based experiments.

I. INTRODUCTION

The Receive Livelock (RL) [1] problem, which is caused under heavy incoming traffic load in hosts containing Gigabit Ethernet (GbE) network interface cards (NIC), prevents the hosts from benefiting from the higher speed supported by the interfaces [2]. Interrupt coalescing (IC) technique is a technique that can be used in general-purpose OSes to mitigate the RL problem. Many of the new GbE NICs have been manufactured to support interrupt coalescing features and most GbE NIC drivers provide interfaces for manually changing interrupt generating rates. In addition, some NIC drivers have implemented adaptive IC schemes. However, all the IC schemes designed so far are based on heuristics. However, IC schemes designed using an *ad-hoc* local-information encounter some problems, namely, (i) there is no way to know if the system performance can be improved; (2) very little is understood about the reasons why a scheme can or cannot improve system performance.

The authors in [3] and [4] modeled the packet reception process as a Markov process and then analyzed interrupt disable-enable scheme and IC technique, respectively, based on queue theory. Some authors such as in [5] and [6] systematically analyzed NAPI [7] in Linux 2.6 kernel, an implementation of the device hybrid interrupt-polling scheme proposed in [8]. These models and analysis either cannot capture the characteristics of IC technique or cannot be directly applied to investigate IC schemes in terms of *system goodput* and CPU consumption and latency over a wide range of hardware and traffic conditions. Note that we use *system goodput* and *goodput* interchangeably in the subsequent sections of this paper.

This paper considers the IC techniques that are implemented in the current general-purpose OSes, in which interrupt handling has absolute priority over those of all other

tasks, on commodity off-the-shelf PCs. We first present an analytical model and then study the performance of IC technique in terms of system *goodput* and CPU consumption and latency. The analysis gives insight into the behavior of the IC techniques.

Considering that the networking subsystems are implemented differently in different OSes and in different versions of the same OS, we describe our model and analysis in the context of Linux kernel 2.6.20. In addition, there exists some difference in the driver implementation of different NICs. Unless otherwise specified, we describe our analysis in the context of Intel Pro/1000 GbE NIC drivers, namely the e1000 driver.

The rest of this paper is organized as follows. Section II presents background. In Section III we present the model and analysis for IC technique and the existing IC schemes. Section IV gives the results of a series of performance tests. Section V discusses the conclusions and describes future work.

II. BACKGROUND

In this section, we first describe how a purely interrupt-driven networking subsystem processes the incoming packets. Then we describe interrupt coalescing.

A. Description of a Purely Interrupt-driven Networking Subsystem

From the software point of view, Fig. 1 describes the process of a packet traversing from the DMA ring (in RAM) to the intended recipient in a purely interrupt-driven networking subsystem. An interrupt is generated right after an incoming packet is put into the DMA ring or a packet is sent out. When the CPU receives the interrupt signal due to an incoming packet, it invokes the RX interrupt service routine (ISR) to remove packets from the DMA ring and put into a temporary queue. In order to improve performance, an ISR invocation can handle more than one packet in the DMA ring but the number of handled packets, denoted by B_i , is limited in order to avoid a long interrupt handling.

Fig. 1 Packet Reception Process in the purely interrupt-driven OS

If no interrupt signal arrives right after the RX ISR execution, the RX softirq service routine (SSR) is invoked for processing the packets by the protocol stack by taking packets from the temporary queue and putting them into *rcv_buffer*. In

The work described in this paper is supported in part by Beijing Jiaotong University Science Foundation under 2007RC033, HK RGC under HKUST6177/04E, National High Technology Research and Development Program of China under 2007AA01Z410.

the rest of the paper, by interrupt we mean a hardware interrupt. Protocol processing typically involves IP protocol processing and transport protocol processing such as TCP, UDP and SCTP. Part of SSR is implemented in the NIC driver. The protocol processing is non-preemptible but can be interrupted. In Linux 2.6, the RX SSR execution is stopped when it passes a given limit of execution time or passes a given limit of the number of processed frames or other interrupt occurs such as timer interrupt. Without loss of generality, we disable the first condition in the following. We use *Budget* (B_p) to denote the maximum number of packets processed in an RX softirq invocation. Its value is fixed in Linux 2.6 kernel. Note that it is possible that RX SSR is interrupted during its execution and continues executing after the interrupt handling. This situation is not an RX softirq invocation, just continuing the last invocation.

If there is still no interrupt signal arrival or if there is no packet in DMA ring for protocol processing or the protocol processing of B_p packets is complete, the application-level processing begins. Due to the priorities of ISR and SSR over those of the normal applications, it is possible that less CPU cycles are left over for the applications under heavy traffic load. In these circumstances a receive livelock occurs.

rcv_buffer has different meanings in different contexts. If the computer is for packet forwarding, *rcv_buffer* denotes the incoming buffer of another NIC. 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 in the rest of the paper.

B. Description of Interrupt Coalescing

Interrupt coalescing is a hardware technique, which groups multiple packets in a single interrupt to limit the rate of interrupt generation by the NIC. Thus, instead of generating an interrupt for every incoming packet, the NIC waits for a predefined number of packets to accumulate in the incoming queue before generating an interrupt. There are five kinds of timers in Intel® GbE Controllers [9] for IC. The most commonly used timer is the Interrupt throttling timer (ITT), which is a simple countdown timer and is supported in most GbE NICs. ITT can work independently of any interrupt source and is not affected by any network events [9]. Fig. 2 (adapted from [9]) describes how the NIC controller generates interrupts when ITT is enabled. The GbE NIC controller blocks all interrupt sources until ITT expires. If interrupt events are pending when ITT expires, the GbE controller generates an interrupt. When the countdown reaches zero, ITT resets and restarts its countdown.

Fig. 2 Interrupt Throttling Timer

III. MODELING AND ANALYSIS OF INTERRUPT COALESCING

The Receive Livelock problem occurs due to the large amount of packet processing overhead incurred before the packet arrives at the recipient. Removing the temporary queue

interrupt; by softirq we mean a software as in NAPI can reduce some overhead. Without loss of generality, we consider a networking subsystem, in which: (i) the RX ISR does nothing more than posting a softirq; (ii) the RX SSR takes packets from the DMA ring and puts into *rcv_buffer*. This is slightly different from the networking subsystem described in Section II. In the section we first present a model for IC and then its analysis. At last we evaluate some existing IC schemes.

Before continuing, we list some assumptions that are used in the subsequent analysis: (i) all the arriving packets have the same packet size and arrive at a constant rate; (ii) *rcv_buffer* size $>$ DMA ring size ($B_{DMA} > B_p$); (iii) there is only one logic CPU; (iv) the effect of timer interrupt is ignored. Ignoring timer interrupt does not affect our results, but simplifies further discussion. Some variables which are used in this section are defined below:

Notation	Description
T	the time interval between two consecutive interrupts generated due to ITT expiration.
T_p	the CPU time for protocol processing a packet
T_a	the CPU time for user-space application processing a packet, including the cost of system call; when cache miss occur, T_a also includes CPU cost of cache missing
T_{ir}	the CPU time for an ISR execution
T_c^a	the CPU time for context switching the user-space application
T_c^i	CPU time from CPU receiving an interrupt signal to the start of ISR execution
λ	the packet arrival rate (pps) at the DMA ring, thus when PCI bus is not a throughput 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 during two consecutive ITT interrupt signals, not more than k_p .
γ	the number of packets in the DMA ring at the beginning of RX SSR invocation
T_f	CPU idle time
G	system goodput

A. Modeling and Analysis of IC

The packet interarrival time, $1/\lambda$, can be in $(T_p, +\infty)$ or $(0, T_p]$. In the second case, if B_i and B_p are infinite, there are no CPU cycles left for application processing no matter how to adjust T due to the priority setting policy and the scheduling policy [10] in Linux 2.6. In this subsection, we only consider the case of $(T_p, +\infty)$. We will discuss the second case in next subsection.

The packet processing time line can be depicted as shown in Fig. 3, in which a round denoted by T is the time interval between the two consecutive interrupt signals, which occurs when CPU is not in protocol processing. If the CPU is doing protocol processing at the time of an ITT interrupt signal arriving, the protocol processing will be stopped and will continue after RX ISR is done. Application processing occurs only when there is no packet in DMA ring. Let $T_o = T_c^i + T_{ir}$. Thus, $T_o + T_c^a + k_a T_a < T$ and $\gamma < T\lambda$ and $T = \eta T_f + \eta T_o + T_c^a + k_p T_p + k_a T_a$, where $\eta \geq 1$.

T_o is fixed for a specific environment, determined by the hardware and software configurations. In a common PC architecture, the data to be processed by the CPU must be copied to page cache and L2 cache, if it is not in each cache. We refer to as memory-jump consumption the CPU cycles consumed in this copying process, during which CPU is idle and cannot do other things in the uniprocessor system. The

number of CPU cycle consumed for any packet from the point of arriving at the NIC to the point of delivery to the application consists of two parts: memory-jump consumption and code execution consumption. Code execution consumption in the protocol processing is same for any incoming packet. So is memory jump consumption. The reason is that any packet from the DMA ring is a new packet to the CPU and thus must be transferred to the L2 cache through a cache miss. Thus, T_p can be regarded as fixed.

Fig. 3 Packet processing time line

Although code execution consumption is same for any packet in the application processing, memory-jump consumption may be varying in the application processing when k_p is very large. Part of the reason is as follows. It is possible that part or all of information of the packet required for application processing has already been put in L2 cache. When k_p is not large, this information is still in L2 cache at the beginning of the application processing. Then there are less cache misses. But if k_p is large, it is possible that the L2 cache area occupied by this information has been reused by other packet. This usually occurs under heavy small-packet traffic load. Our experiment result shows that the effect of the T_a change on G is not significant. The following analysis assumes that T_a is also unchanged.

In the following, we use I^m to denote I , η^m to denote η , T^m to denote T and k_p^m to denote k_p in the m^{th} round. The problem of maximizing the system goodput can be formally specified as:

$$\text{maximize } G = k_a / (\eta T) \quad (1)$$

$$\text{subject } \left\{ \begin{array}{l} k_a = \min \left\{ \frac{(\eta T - k_p T_p - \eta T_o - T_c^a)}{T_a}, k_p \right\} \\ (\eta - 1)(T - T_o) \leq k_p T_p < \eta(T - T_o) \end{array} \right. \quad (2)$$

$$\quad (3)$$

Proposition 1. For a given T , $G > 0$ when $1/\lambda > T_p$.

Proof. $k_p T_p = \frac{\eta T_p}{1 - T_p \lambda} < \frac{T \lambda T_p}{1 - T_p \lambda}$ (by $1/\lambda > T_p$ and $\gamma < T \lambda$). Thus,

$$k_p T_p < T \left\lceil \frac{\lambda T_p}{1 - T_p \lambda} \right\rceil. \text{ Let } \eta = \left\lceil \frac{\lambda T_p}{1 - T_p \lambda} \right\rceil. \text{ Thus, an RX SSR}$$

invocation must stop after ηT time when $1/\lambda > T_p$. Thus, there are CPU cycles left for application processing. ■

Proposition 2. If PCI bus is not a bottleneck and

$$B_{DMA} > B_p > \left\lceil \frac{T \lambda}{1 - T_p \lambda} \right\rceil, \text{ there is no packet dropped in the NIC.}$$

Proof. Assume there is packet dropped in the NIC. The description in Section II and the above assumption indicate that there is no new packet in the DMA ring at the beginning of the application-level processing. Thus, dropping must occur

in the protocol processing (by $B_{DMA} > B_p > \left\lceil \frac{T \lambda}{1 - T_p \lambda} \right\rceil$ and $\gamma < T \lambda$).

DMA ring must overflow in the protocol processing (by the assumption that PCI bus is not a bottleneck). Thus,

$k_p = \frac{\gamma}{1 - T_p \lambda} > B_{DMA} > \left\lceil \frac{T \lambda}{1 - T_p \lambda} \right\rceil$ and then $\gamma > T \lambda$. There is a contradiction. ■

Proposition 3. Assume PCI bus is not a bottleneck and $B_{DMA} > B_p > \left\lceil \frac{T \lambda}{1 - T_p \lambda} \right\rceil$. $G = \lambda$ if and only if rcv_buffer does not overflow.

Proof. rcv_buffer does not overflow $\Leftrightarrow$ All the incoming packets are received by the application. (by Proposition 2) $\Leftrightarrow G = \lambda$. ■

Proposition 4. Assume PCI bus is not a bottleneck and $B_{DMA} > B_p > \left\lceil \frac{T \lambda}{1 - T_p \lambda} \right\rceil$. $G = \lambda$ if and only if $\eta T = T_f + \eta T_o + T_c^a + \eta T \lambda (T_p + T_a)$ and $T_o + T_c^a + \eta T \lambda T_a < T$.

Proof. $G = \lambda \Leftrightarrow k_p = k_a$ and $G = k_a / \eta T = \lambda$ and $T_c^a + T_o + \eta T \lambda T_a < T \Leftrightarrow k_p = k_a = \eta T \lambda$ and $T_c^a + T_o + \eta T \lambda T_a < T \Leftrightarrow \eta T = T_f + \eta T_o + T_c^a + \eta T \lambda (T_p + T_a)$ and $T_c^a + T_o + \eta T \lambda T_a < T$. ■

Proposition 5. Assume (i) $I^1 = \eta^1 T^1 > I^2 = \eta^2 T^2$ and $T^1 > T^2$; (ii) both I^1 and I^2 can lead to $G = \lambda$. Then the CPU consumption for processing each incoming packet under I^1 is less than under I^2 .

Proof. There exist T_f^1 and T_f^2 such that $I^1 = T_f^1 + \eta^1 T_o + T_c^a + (I^1 \lambda) T_a + (I^1 \lambda) T_p$ and $I^2 = T_f^2 + \eta^2 T_o + T_c^a + (I^2 \lambda) T_a + (I^2 \lambda) T_p$ (by Proposition 4). CPU consumption for processing a packet is $T_o / (T \lambda) + T_c^a / (I \lambda) + T_a + T_p$. Thus, the CPU time for processing an incoming packet under I^1 is less than under I^2 . For a given traffic load, the packet arriving rate is constant. Thus the conclusion. ■

The less CPU consumption means that (i) more power is saved; (ii) CPU can handle more other applications.

Proposition 6. For given λ , if $\frac{\tau-1}{\tau} < T_p \lambda < \frac{\tau}{\tau+1}$ where τ is non-negative integer, then the upper bound of η is τ and γ in a round determines η in the following round.

Proof. Assume $\gamma = \varepsilon T \lambda$ and $k_p T_p = (\varepsilon T \lambda) / (1 - T_p \lambda) = x T / T_p$, where $0 < \varepsilon < 1$ and x is non-negative. Thus, $\frac{\tau-1}{\tau} < \frac{x}{x+\varepsilon} < \frac{\tau}{\tau+1}$ (by $\frac{\tau-1}{\tau} < T_p \lambda < \frac{\tau}{\tau+1}$). Thus, $(\tau-1)\varepsilon < x < \tau\varepsilon$. Thus the conclusion. ■

When $\eta > 1$, it is difficult to analyze the relationship between the system performance and T due to the unpredictable η . However, when $1/\lambda > 2T_p$ or $B_p = T \lambda$, the system becomes predictable. In the following we investigate the characteristics of the system under $1/\lambda < 2T_p$ or $B_p = T \lambda$.

Proposition 7. Assume $1/\lambda > 2T_p$. Then $\eta = 1$ and $k_p = T \lambda$ when the system is in the steady state.

Proof. Proposition 6 suggests $\sigma(\tau+1) = 1/2$. Thus $T_o + k_p T_p < T$ and then $\eta = 1$. Assume $\gamma = k_p^0$ at the beginning of the first round.

Thus, $k_p^m = T \lambda + \left(-\frac{T_p \lambda}{1 - T_p \lambda} \right)^m (k_p^0 - T \lambda)$ based on the above

discussions. $\lim_{m \rightarrow \infty} \left(-\frac{T_p \lambda}{1 - T_p \lambda} \right)^m = 0$ (by $1/\lambda > 2T_p$). Thus, $k_p = T\lambda$ when the system is in the steady state. ■

Proposition 8. Assume $B_p = T\lambda$. Then $\eta = 1$ and $k_p = T\lambda$ when the system is in the steady state.

Proof. $T/T_p > T\lambda$ (by $1/\lambda > T_p$ and $B_p = T\lambda \geq k_p$). Thus $T_o + k_p T_p < T$ and then $\eta = 1$. We first prove $k_p = B_p$ when T is unchanged. $T^{m+1} = T^m$ and $k_p^m \leq B_p$ (by k_p definition) and $k_p^{m+1} = \min\{(T^m - T_p k_p^m) \lambda / (1 - T_p \lambda), \lambda T^m\}$. Thus $(T^m - T_p k_p^m) \lambda / (1 - T_p \lambda) \geq T^m \lambda$ and then $k_p^{m+1} = T^m \lambda$. Now we prove $k_p = B_p$ after T is increased in the m^{th} round. No matter whether T is increased in the protocol processing or in the application processing of the m^{th} round, $T^{m+1} < T^m$. Thus $k_p^m = k_p^{m-1} = T^{m-1} \lambda < T^m \lambda$ and $(T^m - T_p k_p^m) \lambda / (1 - T_p \lambda) > T^m \lambda$. But $k_p^{m+1} = \min\{(T^m - T_p k_p^m) \lambda / (1 - T_p \lambda), T^{m+1} \lambda\}$. Thus, $k_p^{m+1} = T^{m+1} \lambda = T^m \lambda$. ■

Proposition 9. Assume (i) $1/\lambda > 2T_p$ or $B_p = T\lambda$; (ii) $G = \lambda$ when $T = T'$. Then (i) $G = \lambda$ for all $T \geq T'$; (ii) The larger T the less CPU consumption for processing an incoming packet for all $T > T'$; (iii) The larger T the larger G for all $T < T'$.

Proof. (i) There exist a non-negative T_f' such that $T = T_f' + T_o + T_c^a + (T'\lambda)T_a + (T'\lambda)T_p$. Let $T \geq T'$. Thus there exist a non-negative T_f such that $T_f = T_f' T/T' + (T_o + T_c^a)(T/T' - 1)$. Thus, $T = T_f + T_o + T_c^a + (T\lambda)T_a + (T\lambda)T_p$. Thus, $G = \lambda$ for all $T \geq T'$. (ii) is a special case of proposition 5. (iii) Assume $T' > T^2$ and both T' and T^2 is smaller than T' . Thus, $G = k_a/T = (T - k_p T_p - T_o - T_c^a)/(T_a T) = (1 - \lambda T_p)/T_a - (T_o + T_c^a)/(T T_a)$. Thus, if $T' > T^2$ then $G' > G^2$. ■

Proposition 9 is based on the assumption that T_a is fixed. As long as the increase in T_a is less than $T_f' T/T' + (T_o + T_c^a)(T/T' - 1)$, Proposition 9 is correct.

Latency for the receiving application when $\eta = 1$: Assume that packets arrive at constant rate. For given T and λ , the average packet latency $= T/2 + (T\lambda + 1)T_a/2 + 1/(2\lambda)$, which is an increasing function of T . Thus, the large T , the large average packet latency.

Latency for other applications when $\eta = 1$: The priority setting policy and the scheduling policy [10] in Linux 2.6 indicate that the protocol processing of k_p packets cannot be preempted by any normal application. Thus, the larger T , the larger k_p and then the higher the maximum latency of other normal applications.

B. Analysis under $1/\lambda < T_p$

In this case G can be improved by adjusting B_p and T . For a given T , B_p is adjusted until rcv_buffer doesnot overflow. The larger T , the larger G when cache missing overhead is ignored.

C. Analysis of some Existing Interrupt Coalescing Schemes

The above analysis indicates:

(1) The effective combination of adjusting B_p and T can produce better performance. The realistic experimental data in [3] shows that it is reasonable to assume $T_p = 2\mu s$ for UDP

traffic in C1 computer. The hardware and software configurations of C1 are given in Section IV. Thus, for the traffic load with Ethernet frame size larger than 500bytes, $1/\lambda > 2T_p$. That is, for such traffic load, G can be improved by increasing T (by proposition 9). The special case ($B_p = T\lambda$) of the situation under $1/\lambda < 2T_p$ also has the features of proposition 9. In order to reduce the overhead caused by computing λ , we could set $B_p = T \times$ the incoming rate of 490-byte traffic. In the following experiments, we use this equation to set B_p when T is changed.

(2) Adjusting T only based on the packet arriving rate (λ) is incomplete (by Eq. (2) and (3)), ignoring the application workload. The dynamical adjusting scheme in E1000 7.3.x driver is such scheme. It defines three interrupt generating rates based on experiments and decides which rate is used according to the incoming traffic pattern in the last timeframe. In order to avoid receive livelock, the rate is always set for the worst case scenarios and then other performance such as system response is degraded.

(3) Proposition 3 shows it is right to adjust T based on whether rcv_buffer overflows. However, how to apply this information is important in uni-processor systems, where the providing and consuming processes of the socket buffer packets are asynchronous. The authors in [11] proposed adjusting the maximum interrupt rate based on the socket buffer utilization periodically. It is possible that the buffer utilization is zero at the end of the consuming process and 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 wrong adjustment is done, leading to the *goodput* fluctuation. The authors in [6] proposed adaptive IC scheme, which adjusts T based on the maximum rcv_buffer utilization in the sampling interval and then avoids the *goodput* fluctuation.

IV. PERFORMANCE EVALUATION

Our experimental platform, shown in Fig. 4, consists of two end systems, labeled C1 and C2. They are connected by a Gigabit switch. The configurations are given in Table I. The PCI buses of C1 and C2 are not bottleneck.

Fig. 4 Experimental setup

Table 1 Computer configurations

Specification	C1	C2
CPU	PD 820/2.8GHz	P4 531/3.0GHz
Connection to chipsets	dedicated bus: 32bit/ 66MHz	dedicated bus: 32bit/66MHz
L2 cache	1024KB	1024KB
RAM	1GB	512MB
Gigabit NIC	Intel 82571	Intel 82571
Driver version	E1000 7.3.20-k2	E1000 7.3.20-k2

All computers run Asianux 2.0 [12], whose kernel is upgraded to 2.6.20. Unless otherwise specified, Hyper-Threading (HT) is disabled. To validate our analysis, we implement the networking subsystem mentioned in Section III by enabling NAPI and removing the operation of disabling RX interrupt. C2 is used as the packet-generator, sending out as many packets as possible such that the full load to C1 can be sustained. All the traffic is UDP/IP based, in order to avoid the effect on the packet generating rate of the flow control and congestion avoidance algorithms defined in TCP protocol.

There is only an application running in C1. To emulate the packet application processing such as storing, the application in C1 performs 200 floating-point multiplications before dropping the received packet.

In one softirq invocation in Linux kernel 2.6.20, each SSR is executed `MAX_SOFTIRQ_RESTART` (set to 10 by default) times and at most `netdev_budget` packets are protocol processed in an SSR execution. In the following experiments, we set `MAX_SOFTIRQ_RESTART=1` and then $B_p = \text{netdev_budget}$. We set `netdev_max_backlog=50000` to avoid its overflow. In addition, the kernel stops the protocol processing by default when the protocol processing time is beyond 1ms. We remove this limitation. Other parameters are set as follows: `rcv_buffer=8000000bytes`, # of DMA ring count =4096.

The performance metrics are *Goodput*, CPU consumption and latency. *Goodput* is defined as the rate at which packets are successfully delivered to and processed by the intended recipients. We compute *goodput* according to the packet size, which is the byte-count in the length field of the IP header. CPU consumption is evaluated by CPU idle percent. Latency is evaluated by Round Trip Time (RTT) of ICMP packets.

A. Effect of T on G and on CPU Consumption

The experiments of this section aim to investigate the effect T on G in C1 computer under different packet sizes. We do experiments by varying maximum interrupt rate from per interrupt per incoming packet (PIPT) to 2000 per second. We experiment six different packet sizes, viz. 64, 128, 256, 512, 1024, and 1500 bytes. Fig. 5 shows the *goodput* versus $1/T$ under different packet sizes. P64 denotes the results when packet size is 64. So are for P128, P256, P512, P1024, and P1500. Fig. 6 plots the CPU idle time percent (measured with `mpstat`) versus $1/T$ of packet size 1500, 1024 and 512 bytes. When packet size is 64, 128 and 256, CPU is always busy. Thus, the results of these packet sizes are not given. The results confirm proposition 9. We observe that G decreases when $1/T$ is small for small packet traffic. It may be due to cache miss.

B. Ping Latency

The experiments in this subsection investigate the effect of T on system latency of C1. 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 GbE NIC. Fig. 7 shows the ping latency variation over time. We observe that the larger T the larger fluctuation in ping latency and the larger average latency. When the ICMP packet arrives just before the protocol processing, the latency is the lowest.

V. CONCLUSIONS AND FUTURE WORK

In this paper we present a model to analyze IC technique in terms of *goodput*, CPU consumption and latency. The analysis gives insight into understanding IC technique. In addition, the analysis provides guidelines about designing efficient NIC drivers and manually adjusting the networking subsystem parameters in the current general-purpose OSes on commodity off-the-shelf PCs connected to high-speed networks.

Note that the analysis in this paper is only validated through realistic experiments. We plan to implement a simulator in order to validate the analysis in detail. In addition, the work in this paper only analyzes the system behavior in the case of $\eta=1$. We plan to investigate the system behavior under infinite B_p in the future.

REFERENCES

- [1]. K.K. Ramakrishnan, "Performance considerations in designing network interfaces," In Journal of Selected Areas in Communications, Volume 11, Issue 2, Feb. 1993 Page(s):203 - 219.
- [2]. 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. IEEE PACT, Sept. 2005.
- [3]. K. Salah, K. El-Badawi, and F. Haidari, "Performance analysis and comparison of interrupt-handling schemes in gigabit networks," In Elsevier Computer Communications, 2007.
- [4]. K. Salah, "To Coalesce or Not to Coalesce," In International Journal of Electronics and Communications, 2007.
- [5]. S.R. Kundu, S. Pal, K. Basu, and S.K. Das, "Performance Modeling and Evaluation of Polling Device Drivers using Gigabit Network Interface Cards," Technical Report, 2006.
- [6]. X.L. Chang, Jogesh K. Muppala, P.C Zou, and X.K. Li, "A Robust Device Hybrid Scheme to Improve System Performance in Gigabit Ethernet Networks", In Proc. IEEE LCN 2007.
- [7]. J.H. Salim, R. Olsson, and A. Kuznetsov, "Beyond Softnet," In Proc. Linux 2.5 Kernel Developers Summit, San Jose, CA, USA, Mar. 2001.
- [8]. 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.
- [9]. "Interrupt Moderation Using Intel® GbE Controllers," Apr. 2007. <http://www.intel.com/design/network/applnots/ap450.htm>
- [10]. D. P. Bovet, and Marco Cesati, "Understanding the Linux Kernel," Third Edition, O'Reilly & Associates, Inc., 2005.
- [11]. A. Indiresan, A. Mehra, and K.G. Shin, "Receive livelock elimination via intelligent interface backoff," TCL Technical Report, University of Michigan, 1998.
- [12]. <http://www.asianux.com>

Fig. 5 Goodput in C1 versus maximum interrupt rate

Fig. 6 CPU idle time percent in C1 versus maximum interrupt rate

Fig. 7 Ping Latency in C2