

PERFORMANCE TUNING OF GIGABIT NETWORK INTERFACES

Robert Geist, James Martin, James Westall, Venkat Yalla

Department of Computer Science

Clemson University

Clemson, SC 29634-0974

email: {rmg|jmarty|westall|vyalla}@cs.clemson.edu

As Ethernets have attained gigabit speeds, it has become obvious that NIC and device driver designs that work well at slower speeds introduce performance problems at gigabit rates. In this paper we present a systematic evaluation of the effects of frame size and interrupt coalescing strategy on the performance of three benchmark workloads on a gigabit LAN. We show that the use of large frames and interrupt coalescing can produce significant performance benefits, but that the proper degree of interrupt coalescing is strongly dependent upon the characteristics of the workload.

INTRODUCTION

As Ethernet bit rates have increased in speed from 100 million bits per second (Mbps) to one gigabit per second (Gbps) and beyond, performance problems associated with both the traditional size of the maximum transfer unit (MTU) and the practice of consuming one packet per interrupt have been reported in several studies [CHAS01, PRAT01, ZAGA02, HOEF04, GRAY02]. At one Gbps the interarrival time between 1518 byte link level frames is approximately 12 microseconds. If the network interface controller (NIC) generates an interrupt on the arrival of each frame, more than 80,000 interrupts per second must be handled by the CPU. Link, network and transport layer headers must also be processed at the same rate. This load is sufficient to drive all but the very fastest processors into a state described by Mogul and Ramakrishnan [MOGU97] as *receive livelock*. Even if the processor is fast enough to avoid receive livelock, a substantial waste of processing power occurs. Furthermore, no current processors are up to the task of handling the 800,000 interrupts per second that might be encountered on a 10 Gbps interface.

Consequently, a variety of techniques have been suggested as ways to improve throughput and reduce CPU overhead in very high speed Ethernets. These include: offloading the TCP checksum to the NIC; offloading the entire TCP/IP protocol stack to the NIC [MART05]; moving the TCP/IP stack to user space [PRAT01]; the use of polling as an alternative to interrupts [PRAT01, MOGU97]; “zero copy” mechanisms that transfer data directly from the NIC to application buffers [MART05, PRAT01]; support for link level frames larger than 1518 bytes [CHAS01]; and the

use of interrupt coalescing mechanisms [ZAGA02, CHAS01, GRAY02]. All of these mechanisms require some level of support in the operating system, the NIC, or both.

The focus this paper is on the effectiveness of the combined use of large frames and interrupt coalescing. These two mechanisms are particularly attractive because of their relative non-invasiveness. Operating system support for both is fully contained within the typical device driver, and, from a hardware perspective, they are simple enough to implement that they are already available in commodity gigabit NICs.

Large frames

At gigabit speeds, the use of 15K byte frames reduces the upper bound on interrupt arrival rate from 80,000 per second to a manageable 8,000 per second. The associated reduction in the number of link, network, and transport layer protocol headers that must be processed yields additional performance benefits.

Interoperability problems are the principal obstacles to the use of large frames. Although the “Jumbo Frame” [CHAS01], having an MTU of 9000 bytes, has been proposed as a standard for high speed Ethernets, there is no single maximum size frame that is supported by all NICs. Many 100 Mbps NICs and some low end gigabit NICs and switches still do not support large frames at all. Nevertheless, in a homogeneous system such as a blade server where interoperability is not an issue the use of large frames can provide significant improvements to the performance

of applications requiring substantial blade-to-blade network traffic.

Interrupt coalescing

Even when large frames are employed, it is often possible to further reduce the interrupt rate by another order of magnitude via interrupt coalescing. Gigabit NICs often have a mechanism whereby the NIC will delay a programmable amount of time after a frame arrival before raising an interrupt. If a new frame arrival commences during this delay, the interrupt is deferred until the new frame arrival completes. This process is repeated until either no new frame arrival commences during the delay period, some programmable upper bound on total interrupt delay is exceeded, or the number of available receive buffers reaches some low water mark. When the NIC finally generates an interrupt, all of the packets that arrived in the burst may be consumed by the operating system at the cost of a single context switch.

An advantage of interrupt coalescing is that it imposes no obstacles to interoperability in a heterogeneous network. However, unlike the use of large frames, interrupt coalescing provides no reduction in protocol processing overhead. That is, an interrupt coalescing scheme that consumed one hundred 1518 byte frames per interrupt at 1 Gbps bit rate would generate only 800 context switches per second, but it would still be necessary to process over 80,000 sets of protocol headers per second!

In the remainder of this paper we report the results of a systematic study of the effect of frame size and interrupt coalescing on the performance of three benchmark workloads. Our work extends results of previous studies [CHAS01, GRAY02] in which the focus is on the performance of a single TCP connection. Performance is characterized with respect to four measures: sustained throughput in Mbps; system mode CPU time consumed; interrupt rate; and packets processed per interrupt. We show that for good throughput with reasonably low system overhead it is desirable to employ both large frames and interrupt coalescing, but that neither very large frames nor extreme levels of interrupt coalescing are necessary to obtain excellent performance on a one Gbps network.

The remainder of the paper is organized as follows. In section two the testbed and workloads used in the study are described. Results obtained with each benchmark are presented and analyzed in section three. A model characterizing system mode CPU overhead is derived in section four, and concluding remarks follow in section five.

EXPERIMENTAL DESIGN

In this section we describe the testbed upon which the performance measurements were made, the test workloads that were measured, the parameters used in the tests, and the performance measures that were captured.

Testbed Architecture

The network on which the tests were performed is a 265 node Linux Beowulf cluster whose primary use is distributed graphics rendering [GEIS02]. Each node has a single 1.6 GHz Pentium 4 processor, 512MB of memory, and two NICs. The 32 bit wide PCI bus runs at 33 Mhz, making it marginally fast enough to consume a simplex one Gbps transfer. All systems run the RedHat 7.1 distribution of Linux and the 2.4.7 kernel. Nodes are connected via an Extreme Black Diamond 6804 switch which supports two virtual local area networks (VLANs). The primary NIC in each node is a 100 Mbps Intel Pro 100. This NIC is a member of the management VLAN which carries NFS, NIS, and remote login traffic.

The secondary NICs are members of the graphics VLAN which, in normal operation, carries graphics rendering commands and pixel data. Most of these NICs are 100 Mbps 3Com 3C905Cs. The standard Linux device driver for this NIC does not support large frames, but the hardware supports a maximum frame size of 8191 bytes. We have produced a variant of the device driver that supports this maximum.

The remaining 32 nodes are equipped with Intel Pro 1000 gigabit interfaces. Intel refers to the specific variant of the Pro 1000 used in these studies as the 82540EM. It supports TCP (but not UDP) checksum offloading, and this feature was enabled in the TCP study. All of our experiments were conducted on the graphics VLAN at times when no graphics rendering was being performed.

Test workloads

Three different workloads were evaluated. For each workload and each network configuration, ten independent replications of the experiment were conducted. Sample mean and standard deviation were computed for each metric.

The UDP workload

In this test, 10 nodes, each having a 100 Mbps interface, served as traffic sources, and one node having a 1 Gbps interface served as the traffic sink. A single process on each of the ten source nodes sent a simplex flow of UDP packets to a receiving process running on the sink. This is *not* a file

transfer benchmark. The contents of the application layer buffer are not modified between sends. The senders were started simultaneously, and the network monitor process on the receiver is started immediately thereafter.

The duration of each test is two minutes of real time which is sufficient to transfer almost 15 billion bytes of data. To avoid introducing IP layer fragmentation and reassembly, application level reads and writes were set to the current MTU minus the length of IP and UDP headers.

The TCP workload

As with the UDP workload, a fan-in approach with 10 senders and one receiver was used. Each sender transmitted 600 million bytes through a 100 Mbps interface. Application layer reads and writes were held at a constant 16K bytes independent of MTU. The send buffer is not modified between writes. Data collected by the network monitor indicated that no packet drops nor retransmissions occurred.

The NFS workload

These experiments employed four nodes all with gigabit NICs. Three nodes functioned as network file system (NFS) version 2 servers and exported their */usr* directory trees. Three NFS client processes ran on the node on which traffic was monitored. The client processes were started concurrently, and each client read all of the files in one of the three NFS mounted */usr* trees. Each */usr* directory tree contained 1.67 billion bytes of data. Unlike the UDP and TCP workloads, disk access speed is a throughput constraining factor here. Although some versions of NFS now run above TCP, the version 2 code in use on these systems employed UDP as a transport protocol.

Workload parameters

Two parameters were varied during the testing. For the results reported here MTU values of 1500, 3000, 4500, and 6000 were used. All systems participating in any test were configured to use the same MTU. The Linux MTU includes the length of transport and network layer protocol headers but does not reflect the length of link layer headers. Thus a UDP packet with a 20 byte IP header and an 8 byte UDP header has a maximum application layer payload of 1472 bytes. Each link level frame is 18 bytes longer than the specified MTU, and the throughput, which is captured at the NIC, includes these bytes. Thus, for the UDP workload, the application layer throughput with a nominal MTU of 1500 is 1472/1518 of the value reported. The NIC operates in full duplex mode for all tests. In this mode each link level frame carries an eight byte preamble and is followed by a 12

byte interframe gap[HOEF04]. Thus, for a nominal MTU of 1500 bytes, the throughput seen at the NIC is bounded by $10^9 \times 1518/1538 = 987$ Mbps.

The other parameter of interest controls the degree of interrupt coalescing. It is referred to by Intel as the RxAbsIntDelay. Its value, expressed in units of 1.024 microseconds, defines the maximum time an interrupt can be delayed. Other operational parameters of the Pro 1000 were fixed as shown below.

TxIntDelay	1400
TxAbsIntDelay	65000
RxDescriptors	256
TxDescriptors	256
RxIntDelay	1200

Table 1: Pro 1000 configuration

These settings effectively permitted the RxAbsIntDelay to define the level to which interrupts were coalesced and caused all recovery of transmit buffers to occur during the processing of receive interrupts.

Performance measurements

Performance measurements are captured by a modified version of Intel's e1000 device driver. The modifications provide a mechanism whereby an application program may periodically acquire and log atomic binary snapshots of relevant performance measures. Measures used in this study include: the number of link level bytes transmitted and received; the number of packets transmitted and received; the number of interrupts in which receive operations were serviced; the current real time as reflected by the TSC register, a 64 bit register that counts the number of processor cycles since the system was powered on; and the standard Linux *kstat* measures of CPU utilization.

The *kstat* values account for CPU consumption in units known as *jiffies*. They contain the total number of jiffies the CPU has spent executing in application mode, nice mode, and system mode since boot time. The length of a jiffy is configurable, but we use the 10 millisecond jiffy, the default in Linux 2.4 kernels.

RESULTS AND ANALYSIS

We describe in this section the impact of MTU and RxAbsIntDelay on our four measures of system performance. For each measure the results are presented in a graphical form. A separate line is plotted for each MTU with RxAbsIntDelay indexing the x-axis. The minimum value of RxAbsIntDelay is 10 units in all cases. All data points are means of 10 runs and are plotted with vertical error bars of length equal to two standard deviations. For all measures

except system mode CPU time, the standard deviations are negligible.

The first measure is receive interrupt rate. The second, which is approximately the inverse of the first, is the number of packets received per interrupt. While these measures provide direct assessments of the effectiveness of an interrupt management strategy, the other two measure are operationally more important. The third measure is the link layer throughput in bits per second and the fourth is the total system mode CPU time consumed during the run.

The UDP workload

Performance characteristics of the UDP workload are shown in Figures 1, 2, 3, and 4. For small values of Rx-AbsIntDelay, the number of packets received per interrupt may appear anomalous. The time to receive a 6018 byte link level frame is almost 50 microseconds. However, for all MTUs, a 10 unit RxAbsIntDelay yielded 2.000 packets per interrupt with a standard deviation less than 0.0005. Intel does not publish a detailed functional specification for the Pro 1000, but this result would indicate that the absolute interrupt delay timer does not start until a packet has been received, and that any packet reception that is in process when the timer does expire is allowed to complete before the interrupt is raised. For larger values of RxAbsIntDelay the number of packets processed per interrupt grows linearly and tracks the expected value reasonably well. At a delay of 960 (not shown) an average of 80, 41, 27, and 22 packets were processed per interrupt.

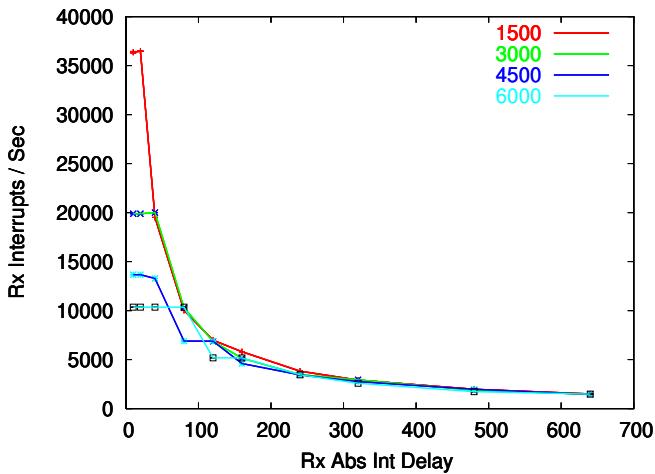


Figure 1: UDP:Rx interrupts per second

Figure 3 shows that throughput in Mbps quickly climbs to its a maximum value. The points, specified as (MTU, Rx-AbsIntDelay, Mbps), at which the throughput curves becomes essentially flat are (1500, 160, 986.7), (3000, 80, 993.1), (4500, 80, 995.3), and (6000, 10, 996.4). When the preamble and interframe gap are factored in, the

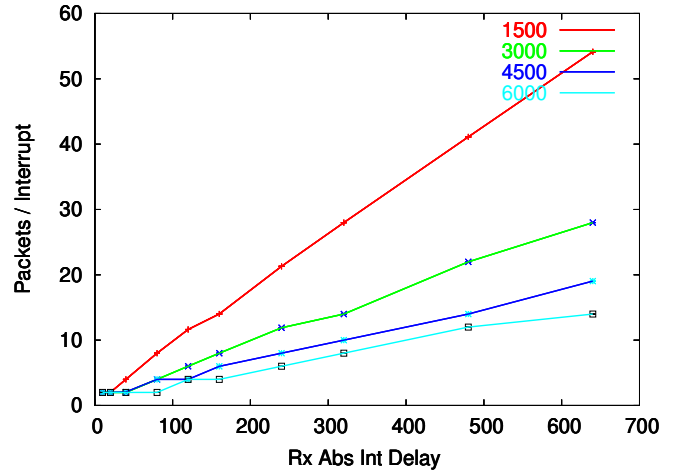


Figure 2: UDP: Packets received per interrupt

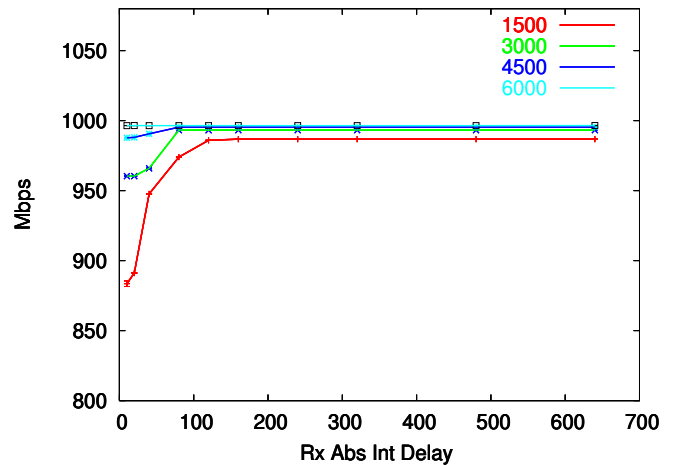


Figure 3: UDP: Throughput in Mbps

maximum possible throughput as measured at the NIC is 996.7 Mbps for a 6000 byte MTU.

Although large values of RxAbsIntDelay yield no measurable throughput gains, they do reduce the interrupt rate and consequently the system mode CPU utilization. This value, shown in figure 4, is the percentage of the elapsed run time during which the CPU is executing in system mode. For the larger MTUs the benefits of further increase beyond 640 appear to be small, but at an MTU of 6000 doubling the delay from 640 to 1280 units yields a significant 15% reduction in system mode CPU time.

The TCP workload

Figures 5 and 6 show that interrupt reduction in the TCP workload is qualitatively similar to that of the UDP workload. The number of packets received per interrupt grows approximately linearly, but at a slightly slower rate than is

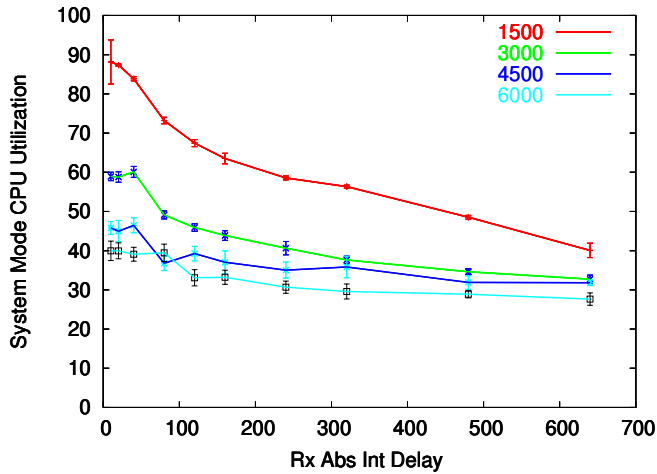


Figure 4: UDP: System mode CPU utilization

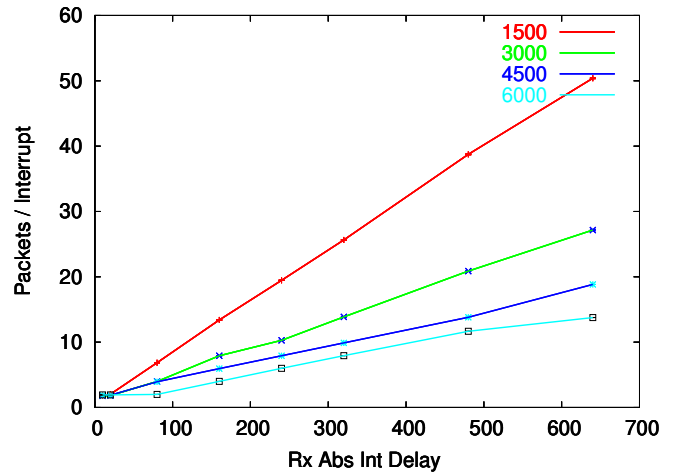


Figure 6: TCP: Packets received per interrupt

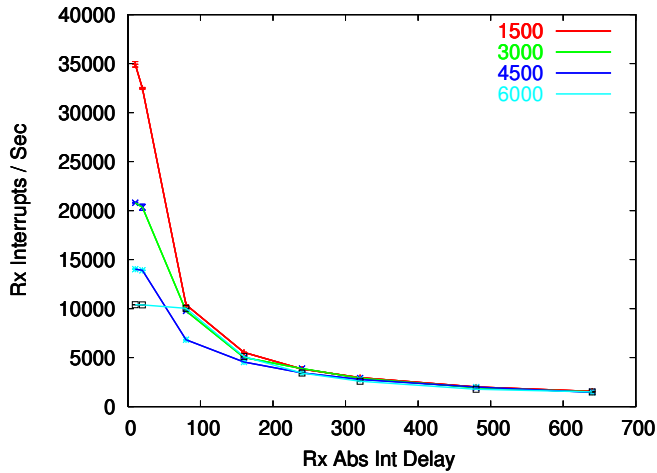


Figure 5: TCP: Rx interrupts per second

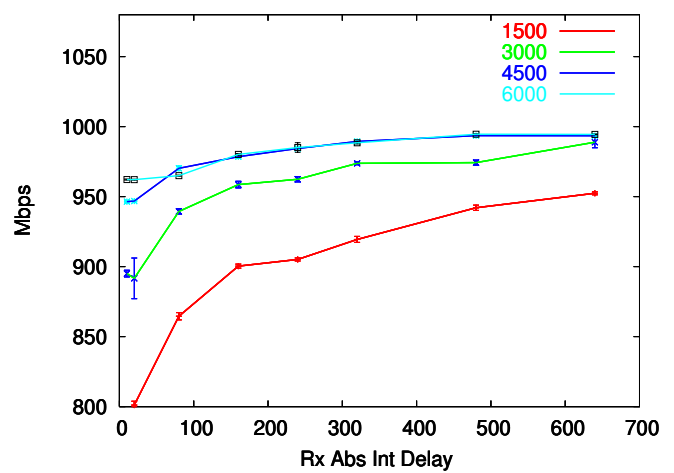


Figure 7: TCP: Throughput in Mbps

the case with UDP. However, unlike UDP the linear growth cannot continue indefinitely. Instead, as the RxAbsIntDelay grows beyond the values shown, the number of packets per interrupt asymptotically approaches an upper bound. This upper bound is a function of the MTU, the number of active connections, and the TCP offered window size. For the 10 connections used in this test, convergence to this upper bound occurs at an RxAbsIntDelay exceeding 3000 units.

As shown in figure 7, significantly larger interrupt delays are needed to achieve maximum throughput with TCP than with UDP. The points at which link layer throughput ceases to grow are: (1500, 960, 970.4); (3000, 800, 991.4); (4500, 480, 993.5); (6000, 480, 994.5). Also unlike UDP, when RxAbsIntDelay becomes large enough to interfere with the timely return of acknowledgments throughput will begin to decay. For an MTU of 6000, this effect was observed at an RxAbsIntDelay of 2000, but, as with the maximum number of packets per interrupt, its onset is a function of MTU,

TCP window size, and the number of active connections.

Figure 8 shows that the system mode CPU utilization, while quantitatively similar to that of the UDP workload, decays more rapidly with the TCP workload. Furthermore, increasing RxAbsIntDelay beyond 480 produces no further benefit of consequence for any MTU. This difference is attributable in part to differences in the applications. The TCP receivers consume 16K bytes per system call regardless of the MTU, but the UDP application consumes only (MTU - 28) bytes per system call. For all MTUs, the UDP application therefore generates more system calls than the TCP workload.

The NFS workload

Based upon the UDP and TCP workloads, one might conclude that the use of an RxAbsIntDelay of at least 640 is indicated. Nevertheless, one might also suspect that an

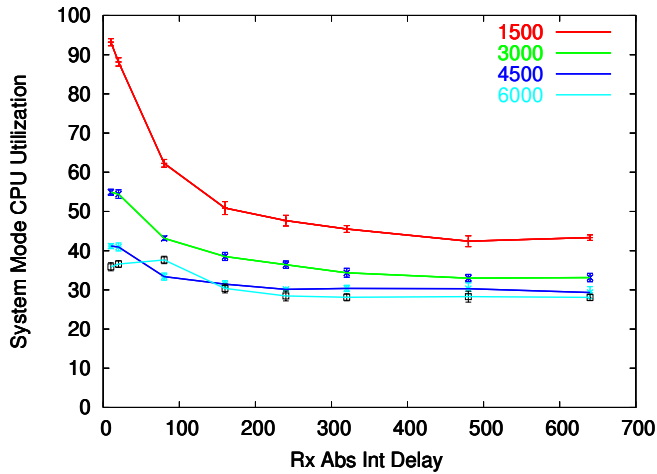


Figure 8: TCP: System mode CPU utilization

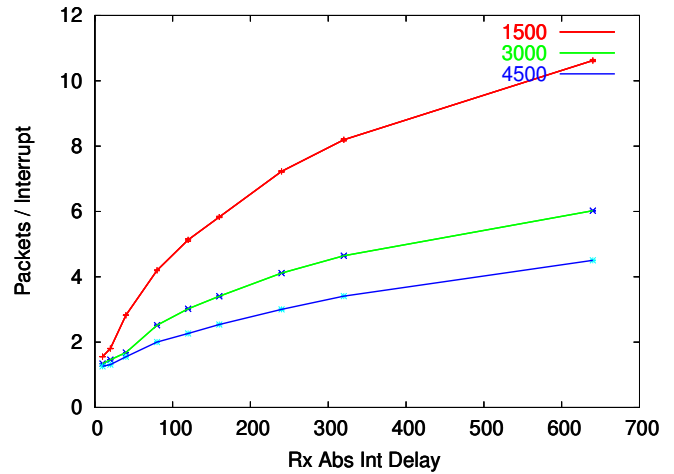


Figure 10: NFS: Packets received per interrupt

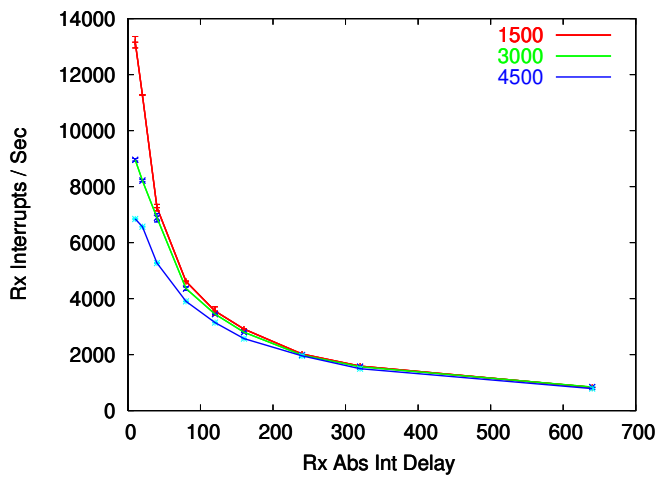


Figure 9: NFS: Rx interrupts per second

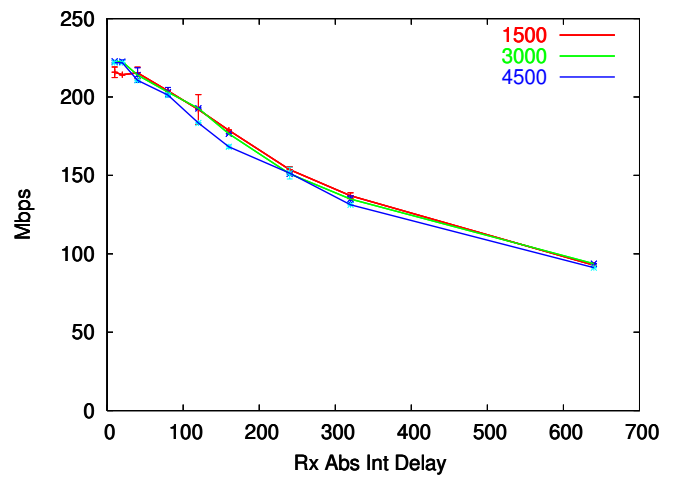


Figure 11: NFS: Throughput in Mbps

aggressive interrupt coalescing strategy might well degrade the performance of a request/response type protocol more strongly than it would impact the performance of a fully pipelined protocol such as TCP. The NFS workload shows that this is indeed the case.

Figures 9 and 10 show the expected benefits of increasing both MTU size and the RxAbsIntDelay. Because the version of NFS in use never sends a packet that exceeds the capacity of a 4500 byte MTU, 4500 is the largest MTU shown. The interrupt rate decays very rapidly on the left side of the graph with the knee of the curve occurring at a value of approximately 160. At a value of 240 all three curves have virtually converged, but the interrupt rate continues to decay reaching a value of slightly more than 800 interrupts per second at an RxIntDelay of 640. The number of packets per interrupt increases in a slightly sub-linear fashion but is clearly still experiencing robust growth at an RxAbsIntDelay of 640.

However, in contrast to the UDP and TCP workloads for which throughput remained nearly constant and optimal over a wide range of increasing values of RxAbsIntDelay, Figure 11 shows that maximum throughput occurs at an RxAbsIntDelay of between 10 and 20 for MTUs of 3000 and 4500 and at a delay of 40 for an MTU of 1500. Beyond these optimal values throughput decays approximately linearly and for all MTUs the throughput achieved at an RxAbsIntDelay of 640 is less than 50% of the maximum.

Figure 12 shows that system mode CPU utilization is still strongly decreasing at an RxAbsIntDelay of 640. This behavior is in marked contrast to that observed in UDP and TCP.

This is also an example of how utilization can be a misleading indicator of performance. The same amount of work is being done by the CPU, but it is simply being amortized over a longer interval because NFS throughput is decaying.

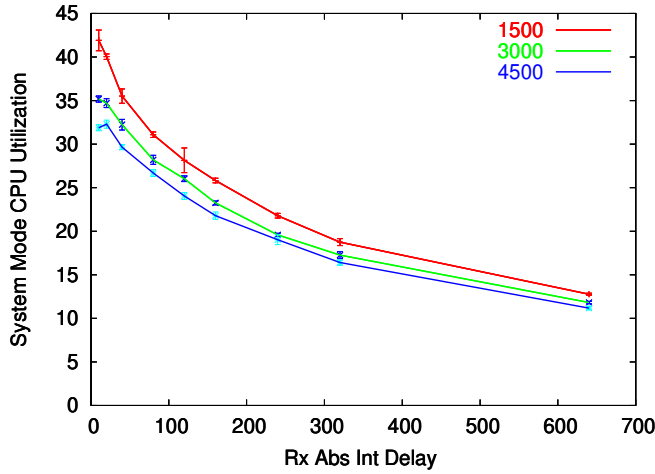


Figure 12: NFS: System mode CPU utilization

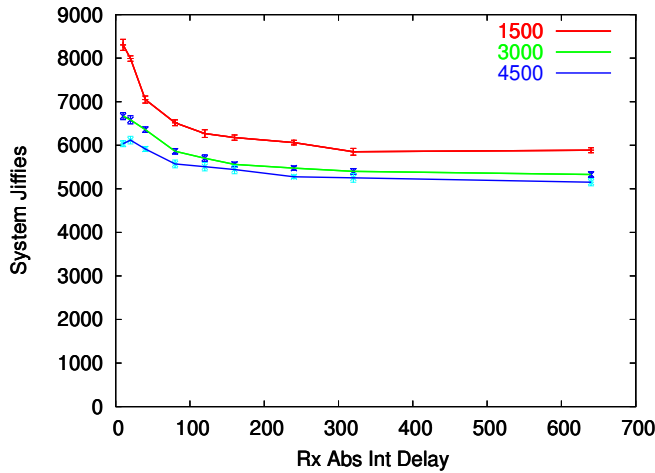


Figure 13: NFS: Aggregate system mode jiffies

Therefore, the total amount of system mode CPU time consumed provides better insight. Figure 13 shows the total number of jiffies consumed for each configuration. Significant reduction in system overhead occurs only until the interrupt delay reaches 80, and little, if any, reduction occurs beyond 320.

Note that, unlike the UDP and TCP workloads, there is no value of RxAbsIntDelay that yields both high throughput and low overhead. If high throughput is the primary objective, an interrupt delay of 40 or less is appropriate, but a value near 100 yields a reasonable balance of throughput and CPU overhead.

CPU OVERHEAD MODEL

In this section we present a simple model for system mode CPU overhead and demonstrate its use on the NFS and TCP workloads. The model distinguishes three sources of

system mode CPU overhead. The per megabyte cost, α , reflects the byte level cost of software checksumming, if required, and copying the data to user space. The per thousand packet cost, β , reflects CPU usage associated with processing individual packets. This includes management of kernel buffers, header processing, and protocol and queue management operations. The cost per thousand interrupts γ reflects the CPU overhead of context switching. The total system mode CPU overhead is then modeled as,

$$T = \alpha N_{MB} + \beta N_{KP} + \gamma N_{KI}$$

where N_{MB} , N_{KP} , and N_{KI} represent the millions of bytes, thousands of packets, and thousands of interrupts associated with a test workload.

The model is clearly a simplification of the real underlying system in that: system mode jiffies are not captured with 100% accuracy; some system mode CPU time that is captured is associated with tasks other than network processing; and the system call overhead produced by the receiver processes is not explicitly modeled. Furthermore, since protocols may differ in the number of times they copy packet data and in per packet processing requirements, it should *not* be expected that a single set of parameters represents the overhead of UDP, TCP, and NFS. Nevertheless, we shall see that a reasonable linear fit is realizable on a per protocol basis.

Model fitting

Each of the test workloads described in the preceding section produces a set of observations,

$$T_m = \alpha N_{MB,m} + \beta N_{KP,m} + \gamma N_{KI,m} \quad \{m = 1, 2, \dots, M\}$$

in which each combination of MTU and RxAbsIntDelay yields a single observation. For each workload, $N_{MB,m}$ is invariant, $N_{KP,m}$ varies with MTU but not RxAbsIntDelay, and T_m and $N_{KI,m}$ vary with both MTU and RxAbsIntDelay. For notational convenience we will express these equations as

$$T_m = \alpha x_m + \beta y_m + \gamma z_m \quad \{m = 1, 2, \dots, M\}$$

The first step in fitting is to normalize the equations so that they all have a common value of T . This is done by multiplying each equation by K/T_m where K is any convenient constant. The next step is to find the mean or centroid of the normalized (x_m, y_m, z_m) ,

$$(c_x, c_y, c_z) = \left(\sum_{m=1}^M x_m, \sum_{m=1}^M y_m, \sum_{m=1}^M z_m \right) / M.$$

The 3×3 covariance matrix $\{m_{i,j}\}$ is given by

$$m_{i,j} = \sum_{m=1}^M v_{m,i} v_{m,j}$$

where

$$\mathbf{v}_m = (x_m - c_x, y_m - c_y, z_m - c_z).$$

It is well known that the eigenvector (e_x, e_y, e_z) corresponding to the smallest eigenvalue of the matrix $\{m_{i,j}\}$ is the normal to the plane that contains the centroid and minimizes the sum of the squares of the distances from the points (x_m, y_m, z_m) to the plane. The normal to the plane

$$K = \alpha x + \beta y + \gamma z$$

is (α, β, γ) . Therefore,

$$(\alpha, \beta, \gamma) = \frac{K}{K_2}(e_x, e_y, e_z)$$

where

$$K_2 = e_x c_x + e_y c_y + e_z c_z.$$

NFS example

Data from the NFS study is shown in table 2 after being normalized using a K value of 10000 jiffies. The first block of values corresponds to an MTU of 1500, the second to 3000, and the third to 4500.

Fitting this data using the procedure described above yields $\alpha = 0.81$ system jiffies per million bytes, $\beta = 0.27$ system jiffies per thousand packets received, and $\gamma = 1.11$ system jiffies per thousand receive interrupts. It can be seen in the table that the maximum deviation of the predicted value from the target value of is 10,000 is 185, a difference of 1.85%.

CPU overhead as a function of MTU and PPI

This model may be reformulated to predict system CPU time as a function of MTU and the number of packets processed per interrupt (PPI). For a billion byte transfer, $N_{MB} = 1000$, $N_{KP} \approx 10^6/MTU$, and $N_{KI} = N_{KP}/PPI \approx 10^6/(MTU \times PPI)$. The predicted overhead in system jiffies per billion bytes transferred in the NFS workload expressed as a function of MTU and PPI is then:

$$T = 1000 \times 0.81 + \frac{10^6}{MTU} \left(0.27 + \frac{1.11}{PPI} \right)$$

This equation reflects facts stated in the introduction. There is a certain fixed cost per megabyte that is unaffected by either interrupt coalescing or frame size. The use of large frames reduces both protocol processing costs and context switching cost. Interrupt coalescing reduces interrupt but not protocol processing costs.

Thus, increasing the MTU has a much more significant impact on the overall cost than does increasing the PPI.

AbsInt Delay	N_{MB}	N_{KP}	N_{KI}	Model Jifs	Delta Jifs
10	6417	4876	3130	9990	-10
20	6668	5067	2807	9886	-114
40	7558	5744	2035	9933	-67
80	8176	6214	1480	9945	-55
120	8501	6460	1259	10030	30
160	8627	6556	1124	10008	8
240	8791	6681	924	9953	-47
320	9108	6921	845	10185	185
640	9052	6879	648	9911	-89
10	7888	3408	2538	10138	138
20	7990	3452	2360	10034	34
40	8274	3575	2131	10044	44
80	8976	3878	1540	10039	39
120	9227	3986	1319	10027	27
160	9463	4088	1201	10115	115
240	9607	4150	1009	10036	36
320	9740	4208	906	10045	45
640	9874	4266	708	9950	-50
10	8676	2694	2141	10147	147
20	8559	2658	2028	9918	-82
40	8853	2749	1772	9896	-104
80	9399	2919	1457	10036	36
120	9506	2952	1303	9961	-39
160	9622	2988	1176	9924	-76
240	9922	3081	1026	10026	26
320	9971	3096	909	9939	-61
640	10166	3157	701	9884	-116

Table 2: Model fitting data

In this example, at an MTU of 1500, the CPU overhead cost per billion bytes is bounded below by 180 jiffies as $PPI \rightarrow \infty$ which is twice the cost obtained with an MTU of 4500 and a PPI of 8.

TCP example

As expected, the TCP transfer yielded somewhat different coefficients with $\alpha = 0.13$ system jiffies per megabyte, $\beta = 0.37$ system jiffies per thousand packets received, and $\gamma = 1.43$ system jiffies per thousand receive interrupts. The large decrease in α is likely attributable to the fact that checksumming was done in hardware for the TCP transfers but must be to be done in software for NFS over UDP.

Figure 14 shows the surface

$$T = 1000 \times 0.13 + \frac{10^6}{MTU} \left(0.35 + \frac{1.41}{PPI} \right)$$

along with the sample data points from the TCP workload. From this diagram, it is clear that for efficient operation frame size must be larger than 1500, and that some level

of interrupt coalescing is very desirable for any frame size. It also shows that frame sizes larger than 6000 and interrupt coalescing factors larger than 8 packets per interrupt provide little additional reduction in CPU overhead.

CONCLUSION

The use of large frames is an absolute necessity if optimal throughput with reasonable CPU loads are to be achieved on a heavily loaded gigabit LAN. With large frames in use, interrupt coalescing can provide additional benefits in both increasing throughput and decreasing CPU overhead, but it can never fully compensate for the penalty associated with using small frames. At low to moderate loads in the range of 200 Mbps, the NFS workload shows that the CPU does not become a throughput constraining bottleneck, but use of a small MTU and small interrupt delay does cause significant amounts of CPU time to be consumed.

For any request/response type protocol, such as the version of NFS used in this study, the window of interrupt delays in which high throughput may be expected is small in size and located in the region of small interrupt delay. Furthermore, it does not overlap the window in which low CPU overhead may be achieved. For these reasons, the use of stop and wait type protocols within gigabit LANs should be avoided where possible.

For any protocol in which packets must be acknowledged, there exists an interrupt delay beyond which throughput will steadily decay. The value of this critical point increases with both the number of active connections and the maximum size of the window of unacknowledged packets. For ten TCP sources on 100 Mbps interfaces the window of interrupt delays in which good performance may be obtained is reasonably wide. However, the NFS workload demonstrates that this result should not be interpreted as a general guideline.

The fact that there is no single configuration that provides optimal performance for all workloads suggests that interrupt delay should be adjusted dynamically depending upon workload characteristics. In fact, Intel's e1000 device driver supports a mode in which this is the case. However, it has been demonstrated that for the 10 source TCP workload a fixed interrupt delay provided significantly higher throughput at lower CPU utilization for all MTUs[YALL05]. This indicates a need for further research into the development of new algorithms for dynamic adjustment of interrupt delay.

ACKNOWLEDGMENT

This work was supported in part by the the ITR Program of the National Science Foundation under award ACI-

0113139.

REFERENCES

- [CHAS01]J. Chase, A. Gallatin, and K. Yocum. End system optimizations for high-speed TCP. *IEEE Communications Magazine*, 39(4):68–74, 2001.
- [GEIS02]Robert Geist, Karl Rache, Rishi Srivatsavai, and James Westall. A distributed rendering system for scientific visualization. In *Proc. 41st Annual ACM Southeast Conference*, pages 359–364, Savannah, Ga, March 2003.
- [GRAY02]Paul Gray and Anthony Betz. Performance evaluation of copper-based gigabit Ethernet interfaces. In *Proc. 27th Annual IEEE Conf. on Local Computer Networks*, pages 679–690, November 2002.
- [HOEF04]T. Hoefler and W. Rehm. A Meta Analysis of Gigabit Ethernet over Copper Solutions for Cluster-Networking. *Chemnitzer Informatik Berichte*, 4(4), 2004.
- [ZAGA02]M. Zec M. Zagar and M. Mikuc. Estimating the impact of interrupt coalescing delays on steady state tcp throughput. *Proc. of the 10th SoftCOM Conference*, 2002.
- [MART05]China Martens. Ethernet start-up wants to be on every server. *Computer World*, June 2005.
- [MOGU97]Jeffrey C. Mogul and K. K. Ramakrishnan. Eliminating receive livelock in an interrupt-driven kernel. *ACM Transactions on Computer Systems*, 15(3):217–252, 1997.
- [PRAT01]Ian Pratt and Keir Fraser. Arsenic: A user-accessible gigabit ethernet interface. In *INFOCOM 2001*, pages 67–76, 2001.
- [YALL05]Venkat R. Yalla. *Tuning Gigabit Interfaces for TCP Performance*. M.S. Thesis, Clemson University, August 2005.

System Jiffies

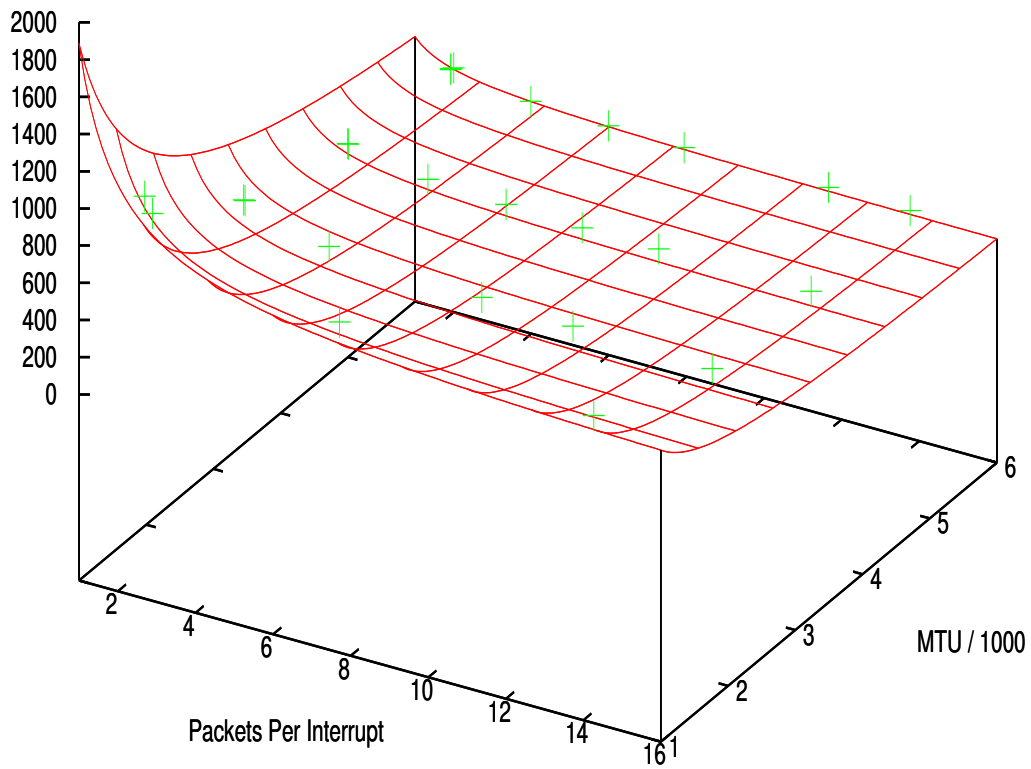


Figure 14: System mode CPU as a function of MTU and PPI