Improving Measured Latency in Linux for Intel® 82575/82576 or X540/82598/82599 Ethernet Controllers

June 2012

Revision 1.2

Contents

Revision History

Rev	Rev Date	Description
1.2	June 2012	Added Intel® X540 Ethernet Controller applicability.
1.1	Nov 2009	Initial public release.
1.0	Oct 2008	Initial release (Intel Confidential).

Note: This page intentionally left blank.

1.0 Improving Measured Latency

1.1 Introduction

The document describes how to improve measured latency with Ethernet networks. Sections include:

- What is latency?
- Why is latency important?
- How latency is measured
- Suggestions for improving latency

This document assumes a Linux system running Fedora Core 11 or a RedHat* Linux.

1.2 What is Latency?

Bandwidth is the main measurement used to describe Ethernet networks. The names used to describe the media directly reflect the throughput: 10/100, 10BaseT, Gigabit Ethernet (GbE), and 10 GbE. Throughput is a good measurement for bulk traffic but traffic can also be composed of many smaller pieces of data that need to be transferred Every small piece of data takes time to make it from the transmitting system, across the network, to the receiving system. That finite time the data takes to make it between these two systems is the latency of the packet. These small times can add up when many small pieces of data are being transferred.

1.3 Why is Latency Important?

Latency affects interactive use of the network. For example, accessing a web site requires multiple requests for text regions, images, menus, and embedded objects. The addition of each of these accesses can result in a visible delay and the latency of each request is a component in that delay.

Inter-server communications are also affected by latency. For example, database accesses are often composed of many individual requests that then spawn more targeted requests. The additive time to send these requests also add up to significant delays. As in the previous example, each request has a latency component that should be minimized.

1.4 How Latency is Measured

The easiest method to measure latency is to use ping and look at the round-trip time for a 56-byte packet (an 84-byte packet including IP headers). This isn't always an ideal measurement because there are internal limitations on ping. The time reported by ping is a round-trip time for the time it takes for the minimal packet to get from the sending CPU down its network stack, across the network, up the receiving network stack to the receiving CPU, and then back through the stacks and network to the originating CPU. One-way times can be calculated by dividing the round trip number in half.

The latency most often measured is the total latency of the network. Most users on the Internet are interested to see how long it takes packets to go from a client computer to a remote server and back. This models the user's experience with the network.

In contrast, our latency measurements target the hardware latency and are performed using NetPIPE from:

http://www.scl.ameslab.gov/netpipe/

NetPIPE performs a ping-pong transfer of various-sized packets to measure latency as well as throughput. The hardware latency is characterized by the transfer time of the least amount of data. The number is calculated by NetPIPE by measuring the round-trip time and then dividing by two.

`netperf` can also be used to test for latency, and allows for latency testing for both TCP and UDP using `tcprr` and `udprr`. `netperf` can also be used to test with different sized packets and can be run repeatedly for statistical analysis. For more information, go to:

http://www.netperf.org/netperf/

1.5 Suggestions for Improving Latency

1.5.1 Turn Off Interrupt Moderation

The largest reduction in latency times are achieved by turning off interrupt moderation. Turning off interrupt moderation might improve small packet latency but trades CPU usage for latency. This causes the driver to take all interrupts and negatively affects Receive Side Coalescing (RSC) and bus efficiency amongst other parameters.

Examples:

The suggested method for varying the interrupt moderation is to use `ethtool` to change the minimum interval between interrupts. `rx-usecs` sets the number of microseconds between interrupts.

```
ethtool -C eth1 rx-usecs 0
```

This can also be changed to be set at startup by editing the scripts in `/etc/sysconfig/network-scripts/ifcfg-eth*`.

The parameters can also be changed during probe time for the 82575EB or 82576EB controllers by editing `/etc/modprobe.conf`, or `/etc/modprobe.d/modprobe-ixbge.conf` or `/etc/modprobe.d/modprobe-igb.conf`.

As an alternative, the parameters can be changed at probe time from the command line:

1. Remove the current igb kernel module (for 82575/82576) or ixgbe kernel module (for X540/82598/82599)

```
rmmod igb
```

or

```
rmmod ixgbe
```

Note: Interfaces might need to be brought down explicitly before the module can be removed.

2. Make sure the module is not present by using `Ismod`.

```
lsmod | grep igb
```

or

```
lsmod | grep ixgbe
```

3. Re-add the igb kernel module with an InterruptThrottleRate of zero. There should be a zero for each interface you are trying to affect. Note that invalid zeroes are ignored and so more comma-separated zeroes on this line are better than fewer.

```
modprobe igb InterruptThrottleRate=0,0,0,0
```

or

```
modprobe ixgbe InterruptThrottleRate=0,0
```

1.5.1.1 Adaptive Moderation

Turning off interrupt moderation delivers the lowest latency but is not practical because of a mixed workload such as a combination of bulk and low latency traffic. In order to balance between CPU efficiency for bulk traffic and low latency, the driver supports adaptive moderation. Adaptive moderation adjusts the interrupt rate dynamically based on packet size and throughput. This can be turned on by setting the Interrupt Throttle Rate (ITR) to zero (1) or three (3).

The value of one (1) implies that it is tuned for dynamic mode and is the only mode supported by the ixgbe driver. The value of three (3) implies dynamic conservative and is supported by the igb driver.

Adaptive moderation is not a substitute for hand-tuning and does not provide the same performance as tuning that is specific to a particular use case.

Examples:

The suggested method for varying the interrupt moderation is to use `ethtool` to change the minimum interval between interrupts. `rx-usecs` sets the number of microseconds between interrupts.

```
ethtool -C ethX rx-usecs 1
```

Setting ITR to zero (0) turns off all interrupt moderation and might improve small packet latency but trades CPU usage for latency. This causes the driver to take all interrupts and negatively affects RSC and bus efficiency as well as other parameters.

The interrupt moderation can also be set at startup by the scripts in `/etc/sysconfig/network-scripts/ifcfg-eth*`.

Optionally, the parameters can also be changed during probe time for the 82575EB or 82576EB controllers by editing `/etc/modprobe.conf` or `/etc/modprobe.d/modprobe-ixbge.conf` and inserting the line:

```
options ixgbe InterruptThrottleRate=1,1,1,1
```

or

```
options igb InterruptThrottleRate=1,1,1,1
```

Additionally, the parameters can be changed at probe time from the command line for the stand alone driver downloaded from e1000.sourceforge.net or the Intel web site (www.intel.com).

```
modprobe ixgbe InterruptThrottleRate=1,1
```

or

```
modprobe igb InterruptThrottleRate=1,1,1,1
```

or

```
modprobe igb InterruptThrottleRate=3,3,3,3
```

For more information about the InterruptThrottleRate parameter, refer to the application note at:

http://www.intel.com/design/network/applnots/ap450.htm

1.5.2 Align Application and Network Interrupt

Additional improvements in latency can be found by pinning interrupts to a specific core and pinning NetPIPE to the same core. Another option is to load the driver with a single Rx/single Tx queue and pin those interrupts to a single CPU.

Another slight gain in latency can be found by pinning all the network interrupts to CPU0. This keeps the interrupt balancing algorithms from spreading out work over the CPUs and increases cache coherency by guaranteeing that most network-related interrupts are assigned to a single CPU.

The ixgbe driver sources contain a script that automatically sets CPU affinities. Stop the irqbalance daemon (`service irqbalance stop`) then run `set_irq_affinity.sh ethX` where `X` is the number of the interface.

Optionally, set the affinities by hand:

1. Make sure IRQ balance is turned off. This fails if the service isn't running, but the error can be ignored.

```
service irqbalance stop
```

2. Make sure the interface is up. The interface must be up for the interrupts to appear in `/proc/interrupts`.
   a. Check `ifconfig -all` to see which interfaces are assigned to the Ethernet controller. This example assumes that there are two interfaces, `eth1` and `eth2`.

```
ifconfig eth1 up
ifconfig eth2 up
```

3. Assign interrupts to specific processors. Check the number of processor (cores) by checking `/proc/cpuinfo`.

```
cat /proc/cpuinfo | grep processor
```

The output for four cores looks like.

```
processor       : 0
processor       : 1
processor       : 2
processor       : 3
```

4. Check the proc file system for interrupts.

Something similar to the following table should appear in `/proc/interrupts`:

```
29    3      0    0    0    PCI-MSI-edge    eth1-tx-0
30    78     0    0    0    PCI-MSI-edge    eth1-rx-0
31    3      0    0    0    PCI-MSI-edge    eth1
32    17     0    0    0    PCI-MSI-edge    eth2-tx-0
33    89     0    0    0    PCI-MSI-edge    eth2-rx-0
34    105    0    0    0    PCI-MSI-edge    eth2
```

The interrupts can be assigned to specific processors by assigning a processor to the interrupts. The processor number is converted to a binary mask, so processor 0 is 1, processor 1 is 2, processor 2 is 4, processor 3 is 8 (2^processor_number).To assign eth1-tx-0 (29) to processor 1, convert the number for the processor (CPU1 = 2^1 = 2) and echo the value to `/proc/irq/29/smp_affinity`.

```
echo 2 > /proc/irq/29/smp_affinity
```

Note: The cores on one socket of a dual-socket system can be assigned non-consecutive numbers. Socket 0 on a dual-socket quad-core system (eight cores) could contain CPU 0, 2, 4, 6 and Socket 1 could contain CPU 1, 3, 5, 7. Check `/proc/cpuinfo` for the *processor*, *physical id*, and *core id*.

Note: RedHat's *RT Affinity Howto* is also a good resource for interrupt binding.

http://rt.et.redhat.com/wiki/index.php?title=RHEL-RT_AffinityHowto&printable=yes

1.5.3 Boot the System Into Single-CPU Mode

A final step is to operate a multi-core system in single processor mode. In this mode, additional reductions in latency times can be achieved, but at a cost. Reducing the system to a single core eliminates the chance for the cache to be invalidated by CPU-to-CPU synchronization; however, all the advantages of multi-core systems are eliminated.

1. Reboot the system and while the grub menu is active, press the "e" key to edit the boot parameters.
2. Use the arrow keys to select the "kernel" line.
3. Press the "e" key to edit the kernel line. At the end of the line, add `maxcpus=1`.
4. Press the "b" key to continue booting.

1.5.4 NUMA

For NUMA, staying within the cache improves performance on 64-bit systems. In addition, running a kernel newer than 2.6.32 and using `numactl` can also help latency.

Note: This page intentionally left blank.