Maximizing Performance of a Gigabit Ethernet NIC Interface

Francesco DiMambro, Sun Microsystems, Inc.

Sun BluePrints™ OnLine—April 2004

http://www.sun.com/blueprints

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95045 U.S.A.
650 960-1300

Part No. 817-6925-10
Revision A, 04/12/04
Edition: April 2004

Please
Recycle

Adobe PostScript

Maximizing Performance of a Gigabit Ethernet NIC Interface

This article describes how to get the greatest benefits from your Sun Gigabit Ethernet network interface card (NIC) interface and a few valuable tools to help you achieve that.

Gigabit Ethernet connections create the greatest stress by far to Sun systems. Therefore, to get the maximum benefit from your gigabit Ethernet NIC interface you need to be aware of the added complications of Auto-negotiation as well as the new ways to ensure that you get the maximum performance from both the gigabit Ethernet interface and the system.

There are two parts to getting the maximum performance from your gigabit Ethernet NIC and the system: first, you need to understand the system itself; second, you need to know the traffic profile through the gigabit Ethernet NIC.

Two key parameters to Sun systems are important for maximizing gigabit Ethernet performance: the number of CPUs in the system and the access time for memory. Establishing the number of CPUs is relatively simple. The memory access time is often hidden, but a simple rule is the larger the system the longer the memory access time. These factors become important for tuning transmitting (Tx) DMA thresholds and deciding how much load balancing of incoming receiving (Rx) traffic is meaningful.

The traffic profile has many dimensions also, including any one of the following characteristics or any combination of them: Rx intensive, Tx intensive or Equal, small packets, large packets, or latency sensitive.

The combination of system parameters and the traffic profile makes it very difficult to enumerate all the possibilities and provide one set of tuning parameters that will address every combination equally and fairly.

Therefore, we can only take the alternative approach of listing the readily available tunable parameters along with an explanation of how and when to use them to get the best results based on your system and application needs.

Each NIC has kernel statistics that provide a means of measuring the traffic profile. You can use this information to adjust `ndd` and `/etc/system` parameters to get the best performance from the NIC.

For more sophisticated features like CPU load balancing, there are some other tools that allow you to look at the system behavior and determine if tuning can better utilize the system as well as the NIC, given the system and the application providing the traffic profile.

- "Network Driver Configuration Parameters" on page 2 describes the details of the three methods you can use for configuring the driver parameters.
- "Ethernet Physical Layer Troubleshooting" on page 5 discusses the physical layer because that layer is the most important with respect to creating the link between two systems.
- "Ethernet Performance Troubleshooting" on page 11 discusses the data link layer, where most problems are performance related.

This article assumes you are an experienced systems administrator, accustomed to working with gigabit Ethernet NIC interfaces.

Network Driver Configuration Parameters

Since this article discusses the network driver configuration parameters it's important to note the details of the network driver configuration methods.

There are three methods you can use for configuring the driver parameters: `ndd`, `driver.conf`, or `/etc/system`.

The `ndd` method is a dynamic form of configuration where you simply invoke the `ndd` command in a command line

```
hostname# ndd -set /dev/ge instance 0
hostname# ndd -set /dev/ge adv_autoneg_cap 1
```

or through an interactive session.

```
hostname# ndd /dev/ge
name to get/set ? instance
value ? 0
name to get/set ? adv_autoneg_cap
value ? 1
name to get/set ?
```

The `ndd` method is excellent for adjusting parameters during normal operation, but the configuration is lost once the system is rebooted. You can avoid this configuration loss by applying the chosen parameter in the `driver.conf` file of the driver you want to configure.

The `driver.conf` file for the device being configured must reside next to the driver being configured in the file system. For example, the Gigabit Ethernet driver has the following path:

```
/kernel/drv/ge
/kernel/drv/ge.conf
```

The following example shows the path for the GigaSwift driver in two different platforms:

```
For Solaris 9 x86
/kernel/drv/ce
/kernel/drv/ce.conf

For Solaris sparc
/platform/sun4u/kernel/drv/ce
/platform/sun4u/kernel/drv/ce.conf
```

Modifying the parameters in the `driver.conf` file can be done with two goals in mind: configuring parameters in a global manner, were all interface instances in the machine using the same driver get the same parameter value, or on a per instance basis, where a parameter value applies to only one instance.

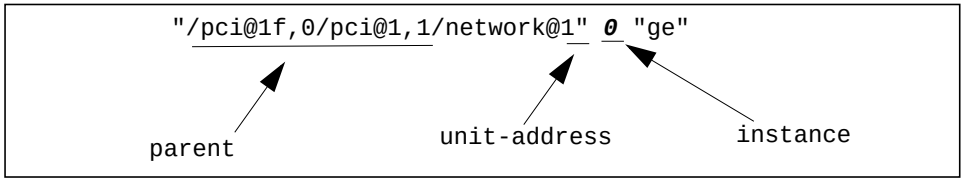
The global configuration method for the `ge.conf` file will appear as follows, applying the `ndd` configuration previously shown:

```
adv_autoneg_cap = 1;
```

Note that the previously shown ndd example only applied instance 1, so the global configuration may be overkill. Determine whether per instance or the global method is more appropriate for your needs.

```
name="ge" parent="/pci@1f,0/pci@1,1"
          unit-address = "1" adv_autoneg_cap = 1;
```

The per instance method does require you to get the ‘parent’ and ‘unit-address’ properties associated with the instance your configuring. This can be achieved by looking at the lines associated with that instance in the path_to_inst file.



The /etc/system configuration method allows you to initialize global variables in the device driver. It has no direct association with ndd and driver.conf setting unless explicitly implemented in the driver. In cases where a driver parameter has been defined for use in either /etc/system or driver.conf, you should choose to use the preferred driver.conf method instead.

Parameters set in the /etc/system always require a system reboot to take effect, the following example shows how an /etc/system variable is set up.

```
hostname# vi /etc/system
...
set ge:ge_intr_mode = 1
....
```

The remainder of the document will discuss ndd, so you should assume that any of the following parameters described as an ndd parameter should only be initialized using the driver.conf file. Any of following parameters described as /etc/system parameters get initialized using only modifications to /etc/system parameters and do require a reboot.

Ethernet Physical Layer Troubleshooting

At the physical layer, failures can prevent the link from coming up. Or worse, the link comes up and the duplex is mismatched, giving rise to less visible problems. The key tool for looking at the physical layer is the `kstat` command. Refer to the `kstat` man page for more information about this useful tool.

The following table lists the general Ethernet MII/GMII kernel statistics and describes their meaning.

TABLE 1 Physical Layer Configuration Properties and Kernel Statistics

Statistic	Values	Description

`cap_autoneg`
`lp_cap_autoneg`
`adv_cap_autoneg`
	0-1	Local interface, advertised and link partner capability.
		0 = Forced mode
		1 = Auto-negotiation

`cap_1000fdx`
`lp_cap_1000fdx`
`adv_cap_1000fdx`
	0-1	Local interface, advertised and link partner capability.
		0 = Not 1000 Mbit/sec full-duplex capable
		1 = 1000 Mbit/sec full-duplex capable

`cap_1000hdx`
`lp_cap_1000hdx`
`adv_cap_1000hdx`
	0-1	Local interface, advertised and link partner capability.
		0 = Not 1000 Mbit/sec half-duplex capable
		1 = 1000 Mbit/sec half-duplex capable

`cap_100T4`
`lp_cap_100T4`
`adv_cap_100T4`
	0-1	Local interface, advertised and link partner capability.
		0 = Not 100-T4 capable
		1 = 100-T4 capable

`cap_100fdx`
`lp_cap_100fdx`
`adv_cap_100fdx`
	0-1	Local interface, advertised and link partner capability.
		0 = Not 100 Mbit/sec full-duplex capable
		1 = 100 Mbit/sec full-duplex capable

`cap_100hdx`
`lp_cap_100hdx`
`adv_cap_100hdx`
	0-1	Local interface, advertised and link partner capability.
		0 = Not 100 Mbit/sec half-duplex capable
		1 = 100 Mbit/sec half-duplex capable

`cap_10fdx`
`lp_cap_10fdx`
`adv_cap_10fdx`
	0-1	Local interface, advertised and link partner capability.
		0 = Not 10 Mbit/sec full-duplex capable
		1 = 10 Mbit/sec full-duplex capable

TABLE 1 Physical Layer Configuration Properties and Kernel Statistics

Statistic	Values	Description

`cap_10fdx`
`lp_cap_10fdx`
`adv_cap_10fdx`
	0-1	Local interface, advertised and link partner capability.
		0 = Not 10 Mbit/sec half-duplex capable
		1 = 10 Mbit/sec half-duplex capable

`cap_asmpause`
`lp_cap_asmpause`
`adv_cap_asmpause`
	0-1	Indicates Local interface, advertised and link partner capability of asymmetric pause Ethernet flow control

`cap_pause`
`lp_cap_pause`
`adv_cap_pause`
	0-1	Indicates Local interface, advertised and link partner capability of symmetric pause Ethernet flow control, when set to 1 and
		`*cap_asmpause` is 0.
		It also has a different meaning based on when `*cap_asmpause` is equal to 1.
		`*cap_pause` = 0 Transmit pauses based on receive congestion.
		`*cap_pause` = 1 Receive pauses and slow down transmit to avoid congestion.

`link_asmpause`	0-1	Indicates the shared link asymmetric pause setting the value is based on local resolution column of Table 37-4 IEEE 802.3 spec. `link_asmpause` = 0 Link is symmetric Pause `link_asmpause` = 1 Link is asymmetric Pause
`link_pause`	0-1	Indicates the shared link pause setting. The value is based on local resolution shown above. If `link_asmpause` = 0 `link_pause` = 0 The link has no flow control. `link_pause` = 1 The link can flow control in both directions. If `link_asmpause` = 1 `link_pause` = 0 Local flow control setting can limit link partner. `link_pause` = 1 Link will flow control local Tx.

TABLE 1 Physical Layer Configuration Properties and Kernel Statistics

Statistic	Values	Description
link_speed	state	The current speed of the network connection in megabits per second
link_duplex	0-2	Indicates the link duplex. link_duplex = 0 Indicates link is down and duplex will be unknown. link_duplex = 1 Indicates link is up and in half duplex mode link_duplex = 2 Indicates link is up and in full duplex mode
link_up	0-1	Indicates whether the link is up or down. link_up = 1 Indicates link is up. link_up = 0 Indicates link is down.

The following example shows a system with ce interfaces present so that some physical layer troubleshooting capabilities can be demonstrated.

```
ifconfig -a
lo0: flags=1000849<UP,LOOPBACK,RUNNING,MULTICAST,IPv4> mtu 8232 index 1
        inet 127.0.0.1 netmask ff000000
ce0: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 2
        inet 67.123.171.236 netmask ffffffe0 broadcast 67.123.171.255
        ether 8:0:20:b1:29:20
ce1: flags=1000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4> mtu 1500 index 2
        inet 67.123.181.236 netmask ffffffe0 broadcast 67.123.181.255
        ether 8:0:20:b1:29:21
```

The first step in checking the physical layer is to check if the link is up.

```
kstat ce:0 | grep link_
     link_asmpause                   0
     link_duplex                     2
     link_pause                      0
     link_speed                      1000
     link_up                         1
```

If the link_up variable is set and a physical connection is present, then things are positive. But also confirm that the speed matches your expectation. For example, if the interface is 1000BASE-TX and you expect the link to run at 1000 Mbit/sec, then

the `link_speed` parameter should indicate 1000. If this is not the case, check the link partner capabilities to see if that is the limiting factor. The following `kstat` command line will show output similar to the following:

```
kstat ce:0 | grep lp_cap
     lp_cap_1000fdx                  1
     lp_cap_1000hdx                  1
     lp_cap_100T4                    1
     lp_cap_100fdx                   1
     lp_cap_100hdx                   1
     lp_cap_10fdx                    1
     lp_cap_10hdx                    1
     lp_cap_asmpause                 0
     lp_cap_autoneg                  1
     lp_cap_pause                    0
```

If the link partner appears to be capable of all the desired speeds, then the problem might be local. There are two possibilities: either the NIC itself is not capable of the desired speed, or the configuration has no shared capabilities that can be agreed on. In either case, the link will not come up. You can check this using the following `kstat` command line.

```
kstat ce:0 | grep cap_
     cap_1000fdx                 1
     cap_1000hdx                 1
     cap_100T4                   1
     cap_100fdx                  1
     cap_100hdx                  1
     cap_10fdx                   1
     cap_10hdx                   1
     cap_asmpause                0
     cap_autoneg                 1
     cap_pause                   0
     .....
```

If all the required capabilities are available for the desired speed and duplex, yet there remains a problem with achieving the desired speed, the only remaining possibility is an incorrect configuration. You can check this by looking at individual `ndd adv_cap_*` parameters, or you can use the `kstat` command:

```
kstat ce:0 | grep adv_cap_
     adv_cap_1000fdx                 1
     adv_cap_1000hdx                 1
     adv_cap_100T4                   1
     adv_cap_100fdx                  1
     adv_cap_100hdx                  1
     adv_cap_10fdx                   1
     adv_cap_10hdx                   1
     adv_cap_asmpause                0
     adv_cap_autoneg                 1
     adv_cap_pause                   0
```

Configuration issues are where most problems lie. All the issues of configuration can be addressed using the `kstat` command above to establish the local and remote configuration, and adjusting the `adv_cap_*` parameters using `ndd` to correct the problem.

The most common configuration problem is duplex mismatch, which is induced when one side of a link is enabled for auto-negotiation and the other is not. This is known as Forced mode and can only be guaranteed for 10/100 Mode operation. For 1000BASE-T UTP Mode operation, the Forced mode (auto-negotiation disabled) capability is not guaranteed because not all vendors support it.

If Auto-negotiation is turned off you must ensure that both ends of the connection are also in Forced mode, and that the speed and duplex are matched perfectly.

If you fail to match Forced mode in gigabit operation, the impact will be that the link will not come up at all. Note that this result is quite different from the 10/100 Mode case. While in 10/100 Mode operation, if only one end of the connection is auto-negotiating (with full capabilities advertised) the link will come up with the correct speed, but the duplex will always be set to half duplex (creating the potential for a duplex mismatch if the forced end is set to full duplex).

If both sides are set to Forced mode and you fail to match speeds, the link will never come up.

If both sides are set to forced mode and you fail to match duplex, the link will come up, but you will have a duplex mismatch.

Duplex mismatch is a silent failure which manifests itself from an upper layer point of view as really poor performance as many of the packets get lost because of collisions and late collisions occurring on the half-duplex end of the connection due to violations of Ethernet protocol induced by the full-duplex end.

The half-duplex end experiences collisions and late collisions while the full-duplex end experiences a whole manner of smashed packets, leading to MIB counters measuring, crc, runts, giants, alignment errors all being incremented.

If the node experiencing poor performance is the half duplex end of the connection, you can look at the `kstat` values for `collisions` and `late_collisions`.

```
kstat ce:0 | grep collisions
     collisions 22332
     late_collisions 15432
```

If the node experiencing poor performance is the full duplex end of the connection, you can look at the packet corruption counters, for example, `crc_err`, `alignment_err`.

```
kstat ce:0 | grep crc_err
     crc_err 22332
kstat ce:0 | grep alignment_err
     alignment_err 224532
```

Depending on the capability of the switch end, or remote end of the connection, there may be an ability to do similar measurements there.

Forced mode while having the problem of creating a potential duplex mismatch also has the drawback of isolating the link partner capabilities from the local station. In Forced mode, you cannot view the `lp_cap*` values and determine the capabilities of the remote link partner locally.

Where possible, use the default of Auto-negotiation with all capabilities advertised and avoid tuning the physical link parameters.

Given the maturity of the Auto-negotiation protocol and its requirement in the 802.3z specification for one gigabit UTP Physical implementations, ensure that Auto-negotiation to enabled.

Deviation from General Ethernet MII/GMII Conventions

There remains some deviation from the general Ethernet MII/GMII kernel statistics which we must address.

In the case of the ge interface, all of the statistics for getting local capabilities and link partner capabilities are read-only ndd properties, so they cannot be read using the kstat command, as described previously, although the debug mechanism is still valid.

To read the corresponding lp_cap_* using ge, use the following commands:

```
hostname# ndd -set /dev/ge instance 0
hostname# ndd -get /dev/ge lp_1000fdx_cap
```

Or you could use the interactive mode, described previously. The mechanism used for enabling Ethernet Flow control on the ge interface is also different, using the parameters in the table below.

TABLE 2 Physical Layer Configuration Properties

Statistic	Values	Description
adv_pauseTX	0-1	Transmit Pause if the Rx buffer is full.
adv_pauseRX	0-1	When you receive a pause slow down Tx.

There's also a deviation in ge for adjusting ndd parameters. For example, when modifying ndd parameters like adv_1000fdx_cap, the changes will not take effect until the adv_autoneg_cap parameter is toggled to change state (from 0-1 or from 1-0). This is a deviation from the General Ethernet MII/GMII convention for the "take affect immediately rule" of ndd.

Ethernet Performance Troubleshooting

Ethernet performance troubleshooting is device specific because not all devices have the same architecture capabilities. Therefore, the discussion of troubleshooting performance issues will have to be tackled on a per-device basis.

The following Solaris™ tools aid in the analysis of performance issues:

- kstat to view device-specific statistics
- mpstat to view system utilization information
- lockstat to show areas of contention

You can use the information from these tools to tune specific parameters. The tuning examples that follow describe where this information is most useful.

You have two options for tuning: using the /etc/system file or the ndd utility.

Using the `/etc/system` file to modify the initial value of the driver variables requires a system reboot for the to take effect.

If you use the `ndd` utility for tuning, the changes take effect immediately. However, any modifications you make using the `ndd` utility will be lost when the system goes down. If you want the `ndd` tuning properties to persist through a reboot, add these properties to the respective `driver.conf` file.

Parameters that have kernel statistics but have no capability to tune for improvement are omitted from this discussion because no troubleshooting capability is provided in those cases.

ge Gigabit Ethernet

The ge interface provides the following tuning parameters that assist in performance troubleshooting.

TABLE 3 ge Performance Tunable Parameters

Parameter	Values	Description
`ge_intr_mode`	0-1	Enables the ge driver to send packets directly to the upper communication layers rather than queueing them. 0 = Packets are not passed in the interrupt service routine but are placed in a streams service queue and passed to the protocol stack later, when the streams service routine runs. 1 = Packets are passed directly to the protocol stack in the interrupt context. Default: 0 (queue packets to upper layers)

TABLE 3 ge Performance Tunable Parameters

Parameter	Values	Description
ge_dmaburst_mode	0-1	Enables infinite burst mode for PCI DMA transactions rather than using cache-line size PCI DMA transfers. This feature supported only on Sun platforms with the UltraSparc® III CPU. 0 = Disabled (default) 1 = Enabled
ge_nos_tmd	32-8192	Number of transmit descriptors used by the driver. Default = 512
ge_put_cfg	0-1	An enumerated type that can have a value of 0 or 1. 0 = receive processing occurs in the worker threads. 1 = receive processing occurs in the streams service queues routine. Default = 1

The ge interface provides some statistics you can use to measure the performance bottlenecks in the driver at the transmit or receive end of the link. The kstats allow you to decide what corrective tuning can be applied, based on the tuning parameters previously described. The useful statistics are shown in TABLE 4.

TABLE 4 List of ge Specific Interface Statistics

kstat name	Type	Description
rx_overflow	counter	Number of times the hardware is unable to receive a packet due to the internal FIFOs being full.
no_free_rx_desc	counter	Number of times the hardware is unable to post a packet because there are no more Rx descriptors available.
no_tmds	counter	Number of times transmit packets are posted on the driver streams queue for processing later by the queue's service routine.
nocanput	counter	Number of times a packet is simply dropped by the driver because the module above the driver cannot accept the packet.
pci_bus_speed	value	The PCI bus speed that drives the card.

When `rx_overflow` is incrementing, packet processing is not keeping up with the packet arrival rate. If it is incrementing and `no_free_rx_desc` is not, this indicates that the PCI bus or SBus bus is presenting an issue to the flow of packets through the device. This could be because the `ge` card is plugged into a slower I/O bus. You can confirm the bus speed by looking at the `pci_bus_speed` statistic. An SBus bus speed of 40 MHz or a PCI bus speed of 33 MHz might not be sufficient to sustain full bidirectional one-gigabit Ethernet traffic.

Another scenario that can lead to `rx_overflow` incrementing on its own is sharing the I/O bus with another device that has similar bandwidth requirements to those of the `ge` card.

These scenarios are hardware limitations. There is no solution for SBus. For PCI bus, a first step in addressing them is to enable infinite burst capability on the PCI bus. You can achieve that by using the `/etc/system` tuning parameter `ge_dmaburst_mode`.

Alternatively, you can reorganize the system to give the `ge` interface a 66-MHz PCI slot, or separate devices that contend for a shared bus segment by giving each of them a bus segment.

The probability that `rx_overflow` incrementing is the only problem is small. Typically, Sun systems have a fast PCI bus, and memory subsystem, so delays are seldom induced at that level. It is more likely is that the protocol stack software might fall behind and lead to the Rx descriptor ring being exhausted of free elements with which to receive more packets. If this happens, then the `kstat no_free_rx_desc` will begin to increment, meaning the CPU cannot absorb the incoming packet in the case of a single CPU. If more than one CPU is available, it is still possible to overwhelm a single CPU. But given that the Rx processing can be split using the alternative Rx data delivery models provided by `ge`, it might be possible to distribute the processing of incoming packets to more than one CPU. You can do this by first ensuring that `ge_intr_mode` is *not* set to 1. Also be sure to tune `ge_put_cfg` to enable the load-balancing worker thread or streams service routine.

Another possible scenario is where the `ge` device is adequately handling the rate of incoming packets, but the upper layer is unable to deal with the packets at that rate. In this case, the `kstat nocanputs` parameter will be incrementing. The tuning that can be applied to this condition is available in the upper layer protocols, although if you're running the Solaris 8 operating system or earlier, then upgrading to the Solaris 9 version will help your application experience fewer `nocanputs`. The upgrade might reduce `nocanput` errors due to improved multithreading and IP scalability performance improvements in the Solaris 9 operating system.

While the Tx side is also subject to an overwhelmed condition, this is less likely than any Rx-side condition. If the Tx side is overwhelmed, it will be visible when the `no_tmds` parameter begins to increment. If the Tx descriptor ring size can be increased, the `/etc/system` tunable parameter `ge_nos_tmd` provides that capability.

ce Gigabit Ethernet

The ce interface provides the following tunable parameters that assist in performance troubleshooting. Note that these are ndd parameters.

TABLE 5 ce Performance Parameters Tunable Using ndd

Parameter	Values	Description
tx-dma-weight	0-3	Determines the multiplication factor for granting credit to the Tx side during a weighted round robin arbitration. Values are 0 to 3. Zero means no extra weighting. The other values are powers of 2 extra weighting, on that traffic. For example, if tx-dma-weight = 0 and rx-dma-weight = 3, then as long as Rx traffic is continuously arriving its priority will be eight times greater than Tx to access the PCI (Default = 0)
rx-dma-weight	0-3	Determines the multiplication factor for granting credit to the Rx side during a weighted round-robin arbitration. Values are 0 to 3. (Default = 0)
infinite-burst	0-1	Allows the infinite burst capability to be utilized. When this is in effect and the system supports infinite burst, the adapter will not free the bus until complete packets are transferred across the bus. Values are 0 or 1. (Default = 0)
red-dv4to6k	0 to 255	Random early detection and packet drop vectors for when FIFO threshold is greater than 4096 bytes and less than 6144 bytes. Probability of drop can be programmed on a 12.5 percent granularity. For example, if bit 0 is set, the first packet out of every eight will be dropped in this region. (Default = 0)

TABLE 5 ce Performance Parameters Tunable Using ndd

Parameter	Values	Description
red-dv6to8k	0 to 255	Random early detection and packet drop vectors for when FIFO threshold is greater than 6144 bytes and less than 8192 bytes. Probability of drop can be programmed on a 12.5 percent granularity. For example, if bit 0 is set, the first packet out of every eight will be dropped in this region. (Default = 0)
red-dv8to10k	0 to 255	Random early detection and packet drop vectors for when FIFO threshold is greater than 8192 bytes and less than 10,240 bytes. Probability of drop can be programmed on a 12.5 percent granularity. For example, if bits 1 and 6 are set, the second and seventh packets out of every eight will be dropped in this region. (Default = 0)
red-dv10to12k	0 to 255	Random early detection and packet drop vectors for when FIFO threshold is greater than 10,240 bytes and less than 12,288 bytes. Probability of drop can be programmed on a 12.5 percent granularity. If bits 2, 4, and 6 are set, then the third, fifth, and seventh packets out of every eight will be dropped in this region. (Default = 0)

TABLE 6 lists the `/etc/system` tunable parameters that assist in performance troubleshooting.

TABLE 6 `ce` Performance Parameters Tunable Using `/etc/system`

Parameter	Values	Description
`ce_ring_size`	32-8192	Determines the multiplication factor for granting credit to the Tx side during a weighted round robin arbitration. Values are 0 to 3. Zero means no extra weighting. The other values are powers of 2 extra weighting, on that traffic. For example, if `tx-dma-weight` = 0 and `rx-dma-weight` = 3, then as long as Rx traffic is continuously arriving its priority will be eight times greater than Tx to access the PCI (Default = 0)
`ce_comp_ring_size`	0-8192	Determines the multiplication factor for granting credit to the Rx side during a weighted round-robin arbitration. Values are 0 to 3. (Default = 0)
`ce_inst_taskqs`	0-64	Allows the infinite burst capability to be utilized. When this is in effect and the system supports infinite burst, the adapter will not free the bus until complete packets are transferred across the bus. Values are 0 or 1. (Default = 0)
`ce_srv_fifo_depth`	30-100000	Random early detection and packet drop vectors for when FIFO threshold is greater than 4096 bytes and less than 6144 bytes. Probability of drop can be programmed on a 12.5 percent granularity. For example, if bit 0 is set, the first packet out of every eight will be dropped in this region. (Default = 0)
`ce_cpu_threshold`	1-1000	Random early detection and packet drop vectors for when FIFO threshold is greater than 6144 bytes and less than 8192 bytes. Probability of drop can be programmed on a 12.5 percent granularity. For example, if bit 0 is set, the first packet out of every eight will be dropped in this region. (Default = 0)

TABLE 6 ce Performance Parameters Tunable Using /etc/system

Parameter	Values	Description
ce_taskq_disable	0-1	Random early detection and packet drop vectors for when FIFO threshold is greater than 8192 bytes and less than 10,240 bytes. Probability of drop can be programmed on a 12.5 percent granularity. For example, if bits 1 and 6 are set, the second and seventh packets out of every eight will be dropped in this region. (Default = 0)
ce_start_cfg	0-1	Random early detection and packet drop vectors for when FIFO threshold is greater than 10,240 bytes and less than 12,288 bytes. Probability of drop can be programmed on a 12.5 percent granularity. If bits 2, 4, and 6 are set, then the third, fifth, and seventh packets out of every eight will be dropped in this region. (Default = 0)
ce_tx_ring_size	0-8192	The mblk size threshold used to decide when to copy a mblk into a pre-mapped buffer, as opposed to using DMA or other methods. Default = 256
ce_no_tx_lb	0-1	The mblk size threshold used to decide when to use the fast path DVMA interface to transmit mblk. Default = 1024
ce_bcopy_thresh	0-8192	The mblk size threshold used to decide when to copy a mblk into a pre-mapped buffer, as opposed to using DMA or other methods. Default = 256
ce_dvma_thresh	0-8192	The mblk size threshold used to decide when to use the fast path DVMA interface to transmit mblk. Default = 1024
ce_dma_stream_thresh	0-8192	This global variable splits the ddi_dma mapping method further by providing Consistent mapping and Streaming mapping. In the Tx direction, for larger transmissions, Streaming is better than Consistent mappings. If the mblk size is greater than 256 bytes but less than 1024 bytes, then mblk fragments will be transmitted using ddi_dma methods. Default = 512

The ce interface provides a far more extensive list of kstats that can be used to measure the performance bottlenecks in the driver in the Tx or the Rx. The kstats allow you to decide what corrective tuning can be applied, based on the tuning parameters described previously. The useful statistics are shown in TABLE 7.

TABLE 7 List of ce Specific Interface Statistics

kstat name	Type	Description
rx_ov_flow	counter	Number of times the hardware is unable to receive a packet due to the internal FIFOs being full.
rx_no_buf	counter	Number of times the hardware is unable to receive a packet due to Rx buffers being unavailable.
rx_no_comp_wb	counter	Number of times the hardware is unable to receive a packet due to no space in the completion ring to post Received packet descriptor.
ipackets_cpuXX	counter	Number of packets being directed to load-balancing thread XX.
mdt_pkts	counter	Number of packets sent using multidata interface.
rx_hdr_pkts	counter	Number of packets arriving which are less than 252 bytes in length.
rx_mtu_pkts	counter	Number of packets arriving which are greater than 252 bytes in length.
rx_jumbo_pkts	counter	Number of packets arriving which are greater than 1522 bytes in length.
rx_ov_flow	counter	Number of times a packet is simply dropped by the driver because the module above the driver cannot accept the packet.
rx_nocanput	counter	Number of times a packet is simply dropped by the driver because the module above the driver cannot accept the packet.
rx_pkts_dropped	counter	Number of packets dropped due to Service FIFO queue being full.
tx_hdr_pkts	counter	Number of packets hitting the small packet transmission method, copy packet into a pre-mapped DMA buffer.
tx_ddi_pkts	counter	Number of packets hitting the mid range DDI DMA transmission method.
tx_dvma_pkts	counter	Number of packets hitting the top range DVMA fast path DMA transmission method.
tx_jumbo_pkts	counter	Number of packets being sent which are greater than 1522 bytes in length.

TABLE 7 List of ce Specific Interface Statistics *(Continued)*

kstat name	Type	Description
tx_max_pend	counter	Measure of the maximum number of packets which was ever queued on a Tx ring.
tx_no_desc	counter	Number of times a packet transmit was attempted and Tx descriptor elements were not available. The packet is postponed until later.
tx_queueX	counter	Number of packets transmitted on a particular queue.
mac_mtu	value	The maximum packet allowed past the MAC.
pci_bus_speed	value	The PCI bus speed that is driving the card.

When rx_ov_flow is incrementing, packet processing is not keeping up with the packet arrival rate. If rx_ov_flow is incrementing while rx_no_buf or rx_no_comp_wb is not, this indicates that the PCI bus is presenting an issue to the flow of packets through the device. This could be because the ce card is plugged into a slower PCI bus. You can confirm the bus speed by looking at the pci_bus_speed statistic. A bus speed of 33 MHz, might not be sufficient to sustain full bidirectional one gigabit Ethernet traffic.

Another scenario that can lead to rx_ov_flow incrementing on its own is sharing the PCI bus with another device that has bandwidth requirements similar to those of the ce card.

These scenarios are hardware limitations. A first step in addressing them is to enable the infinite burst capability on the PCI bus. Use the ndd tuning parameter infinite-burst to achieve that.

Infinite burst will help give ce more bandwidth, but the Tx and Rx of the ce device will still be competing for that PCI bandwidth. Therefore, if the traffic profile shows a bias toward Rx traffic and this condition is leading to rx_ov_flow, you can adjust the bias of PCI transactions in favor of the Rx DMA channel relative to the Tx DMA channel, using ndd parameters rx-dma-weight and tx-dma-weight

Alternatively, you can reorganize the system by giving the ce interface a 66-MHz PCI slot, or separate devices that contend for a shared bus segment by giving each of them a bus segment.

If this doesn't contribute much to reducing the problem, then you should consider using Random Early Detection (RED) to ensure that the impact of dropping packets is minimized with respect to keeping connections alive which would be normally terminated due to regular overflow. The following parameters that allow enabling RED are configurable using ndd: red-dv4to6k, red-dv6to8k, red-dv8to10k, and red-dv10to12k.

The probability that `rx_overflow` incrementing is the only problem is small. Typically Sun systems have a fast PCI bus and memory subsystem, so delays are seldom induced at that level. It is more likely that the protocol stack software might fall behind and lead to the Rx buffers or completion descriptor ring being exhausted of free elements with which to receive more packets. If this happens, then the `kstats` parameters `rx_no_buf` and `rx_no_comp_wb` will begin to increment. This can mean that there's not enough CPU power to absorb the packets but it can also be due to a bad balance of the buffer ring size versus the completion ring size, leading to the `rx_no_comp_wb` incrementing without the `rx_no_buf` incrementing. The default configuration is one buffer to four completion elements. This works great provided that the packets arriving are larger than 256 bytes. If they are not and that traffic dominates, then 32 packets will be packed into a buffer leading to a greater probability that configuration imbalance will occur. For that case, more completion elements need to be made available. This can be addressed using the `/etc/system` tunables `ce_ring_size` to adjust the number of available Rx buffers and `ce_comp_ring_size` to adjust the number of Rx packet completion elements. To understand the traffic profile of the Rx so you can tune these parameters, use `kstat` to look at the distribution of Rx packets across the `rx_hdr_pkts` and `rx_mtu_pkts`.

If `ce` is being run on a single CPU system and `rx_no_buf` and `rx_no_comp_wb` are incrementing, you will have to resort again to RED, or enable Ethernet flow control.

If more than one CPU is available, it is still possible to overwhelm a single CPU. Given that the Rx processing can be split using the alternative Rx data delivery models provided by `ce`, it might be possible to distribute the processing of incoming packets to more than one CPU, described earlier as Rx load balancing. This will happen by default if the system has four or more CPUs, and it will enable four load-balancing worker threads. The threshold of CPUs in the system and the number of load-balancing worker threads enabled can be managed using the `/etc/system` tunables `ce_cpu_threshold` and `ce_inst_taskqs`.

The number of load balancing worker threads, and how evenly the Rx load is being distributed to each worker thread can be viewed with the `ipacket_cpu`*xx* `kstats` the highest number of *xx* tells you how many load balancing worker threads are running while value of these parameters give you the spread of the work across the instantiated load balancing worker threads. This, in turn, gives an indication if the load balancing is yielding a benefit. For example, if all `ipacket_cpu`*xx* have an approximately even number of packets counted on each then the load balancing is optimal. On the other hand, if only one is incrementing and the others are not, then the benefit of Rx load balancing is nullified.

It is also possible to measure whether the system is experiencing a even spread of CPU activity using `mpstat`. In the ideal case, if you experience good load balancing as shown in the `kstats ipackets_cpu`*xx*, it should also be visible in `mpstat` that the workload is evenly distributed to multiple CPUs.

If none of this benefit is visible, then disable the load balancing capability completely, using the `/etc/system` variable `ce_taskq_disable`.

The Rx load balancing provides packet queues, also known as service FIFOs, between the interrupt threads which fan out the workload and the service FIFO worker threads which drain the service FIFO and complete the workload. These service FIFOs are of fixed size, controlled by the `/etc/system` variable `ce_srv_fifo_depth`. It is possible that the service FIFOs can also overflow, and drop packets as the rate of packet arrival exceeds the rate with which the service FIFO draining thread can complete the post processing. These dropped packets can be measured using the `rx_pkts_dropped kstat`. If this is measured as occurring, you can increase the size of the service FIFO, or you can increase the number of service FIFOs allowing more Rx load balancing. In some cases, it may be possible to eliminate increments in `rx_pkts_dropped`, but the problem may move to `rx_nocanputs`, which is generally only addressable by tuning that can be applied by upper layer protocols, although if you're running the Solaris 8 operating system or earlier, then upgrading to the Solaris 9 version will help your application experience fewer `nocanputs`. The upgrade might reduce `nocanput` errors due to improved multithreading and IP scalability performance improvements in the Solaris 9 operating system.

There is a difficulty is maximizing the Rx load balancing, and that's contingent on the Tx ring processing. This is measurable using the `lockstat` command and will show contention on the `ce_start` routine at the top as the most contended driver function. This contention cannot be eliminated, but it is possible to employ a new Tx method known as Transmit serialization, which keeps contention to a minimum while forcing the Tx processes on a fixed set of CPUs. Keeping the Tx process on a fixed CPU reduces the risk of CPUs spinning waiting for other CPUs to complete their Tx activity, ensuring CPUs are always kept busy doing useful work. This transmission method can be enabled using the `/etc/system` variable `ce_start_cfg`, setting it to `1`. When you enable Transmit serialization, you will be trading off Transmit latency for avoiding mutex spins induced by contention.

The Tx side is also subject to an overwhelmed condition, which occurs when the CPU speed exceeds the Ethernet line rate, although this is less likely than any Rx side condition. When the Tx side becomes overwhelmed, `tx_max_pending` value matches the size of the `/etc/system` variable `ce_tx_ring_size`. If this occurs, you know that packets are being postponed because Tx descriptors are being exhausted. Therefore the size of the `ce_tx_ring_size` should be increased.

The `tx_hdr_pkts`, `tx_ddi_pkts`, and `tx_dvma_pkts` are useful for establishing the traffic profile of an application and matching that profile with the capabilities of a system. The parameters `ce_bcopy_thresh`, `ce_dvma_thresh`, and `ce_dma_stream_thresh` are used for adjusting the transmission method applied

to an outgoing packet. These parameters are described in TABLE 7 in terms of mblks, which is the mechanism used to transmit packets in the Solaris operating system. The following output shows how these parameters relate to each other:

```
mblk size < ce_bcopy_thresh: driver will copy into pre-mapped
                                buffer

mblk size > ce_dvma_thresh: driver uses fast path DVMA
                               interface

ce_dma_stream_thresh < mblk size < ce_dvma_thresh:
                                   driver uses streaming DMA method

Otherwise: driver uses consistent DMA method.
```

How to set these parameters is again system dependant and application dependant. The system dependency is associated with memory latency. The rule of thumb to apply here is if the system has a large number of CPUs the memory latency will tend to be larger.

Considering larger memory latency systems it's best to avoid moving data from one memory location to another, so using the premapped buffer for DMA will be more expensive than setting up and tearing down DMA mapping on a per-packet basis.

Furthermore, if the `tx_hdr_pkts` appears to be incrementing at a higher rate than `tx_dvma_pkts`, you have an application with a traffic profile that uses a lot of small packets. Therefore, you should adjust the `ce_dvma_thresh` and `ce_bcopy_thresh` so that most of the packets hit the `tx_dvma_pkts` path in the driver and avoid copies. The following may be reasonable parameters, for such a system:

```
ce_bcopy_thresh = 97
ce_dvma_thresh = 96
ce_dma_stream_thresh = <don't care>
```

Alternatively, in low memory latency systems, the inverse is true and you would need to adjust `ce_dvma_thresh` and `ce_bcopy_thresh` so that most packets take the `bcopy` route.

```
ce_bcopy_thresh = 256
ce_dvma_thresh = 255
ce_dma_stream_thresh = <don't care>
```

The Streaming DMA and Consistent DMA methods are provided as the fall back path, and tend to provide little improvement over the Fast DVMA method or the copy into premapped buffer method. This can be tuned out most of the time, as shown in the previous examples, since it seldom gives improvement over the Fast DVMA method.

You can adjust the DMA thresholds of `ce_bcopy_thresh`, `ce_dvma_thresh`, and `ce_dma_stream_thresh`, using the `/etc/system` file to push more packets into the preprogrammed DMA versus the per-packet programming. Once the tuning is complete, the statistics can be viewed again to see if the tuning took effect.

The `tx_queueX` parameter gives a good indication of whether Tx load balancing is happening. Like the Rx side, if no load balancing is visible, meaning all the packets appear to be getting counted by only one `tx_queue`, then you should switch this feature off and use the `ce_no_tx_lb` variable.

The `mac_mtu` gives an indication of the maximum size of packet that will make it through the `ce` device. It is useful to know if jumbo frames is enabled at the DLPI layer below TCP/IP. If jumbo frames is enabled, then the MTU indicated by `mac_mtu` will be 9216.

This is helpful as it will show that if there's a mismatch between the DLPI layer MTU and the IP layer MTU, allowing troubleshooting to occur in a layered manner.

Once jumbo frames is successfully configured at the driver layer and the TCP/IP layer, then use the `rx_jumbo_pkts` and `tx_jumbo_pkts`, to ensure Transmits and Receives of jumbo frame packets respectively is happening correctly.

Conclusion

Ethernet NIC configuration has become more than simply enabling the interface below TCP/IP or any other Layer 3 and above protocol stacks. The two layers provided by the existing DLPI device driver and hardware need some special understanding to ensure that packets flow, in the first place. Once packet flow is established, you must ensure that packets flow quickly and smoothly to those upper layer protocol stacks.

What defines "quickly and smoothly" is very system specific, in conjunction with being networking application specific. This makes the one-driver-fits-all expectation difficult to achieve, but not impossible, provided the tuning capability is available.

The tuning capability provided lets the driver stretch to fit, and the measurement tools `kstat`, `mpstat`, and `lockstat` help to give a measure of how to mold and shape the driver to achieve the best fit.

Ultimately, the best fit will be set by the customer's expectations of their application and the Sun system that hosts it. We hope this BluePrint article will help customers measure the performance of their drivers, and use those measurements to set the tunable parameters of the gigabit Ethernet drivers to maximize their performance.

About the Author

Francesco DiMambro (frank.dimambro@sun.com), has a Bachelor of Engineering degree with Honors in Information Engineering from Strathclyde University, Scotland and a Master of Engineering degree in Digital Systems Engineering from Heriot-Watt University, Scotland. The first company to utilize his talent beginning 1991 was National Semiconductor, in their Local Area Network division. There he developed NDIS 2 Ethernet network device drivers, which were used in MS-DOS, OS/2 (LAN Manager products) and Windows operating systems. In 1995, he moved from Scotland to California and started NDIS 4 device driver development for Infra Red wireless communication devices operating primarily on Windows 95, 98, and NT.

In 1996, he joined Sun Microsystems, Network Products Group and began supporting the Solaris Ethernet device drivers for Sun. He quickly moved to understanding the Sun Ethernet controllers and the performance factors associated their network device driver implementation. That experience prepared him for the task of developing the next-generation gigabit Ethernet device driver for the GigaSwift Ethernet product line.

Sun Ethernet driver performance has been his primary goal, while also increasing device driver quality and usability. With those areas in mind, he has filed eight patents, and with this first article, marks a new adventure in helping users maximize performance of an Ethernet NIC interface.

Ordering Sun Documents

The SunDocs℠ program provides more than 250 manuals from Sun Microsystems, Inc. If you live in the United States, Canada, Europe, or Japan, you can purchase documentation sets or individual manuals through this program.

Accessing Sun Documentation Online

The docs.sun.com℠ web site enables you to access Sun technical documentation online. You can browse the docs.sun.com archive or search for a specific book title or subject. The URL is http://docs.sun.com/

To reference Sun BluePrints™ OnLine articles, visit the Sun BluePrints OnLine web site at: http://www.sun.com/blueprints/online.html