10 Gb Ethernet

Mark Wagner
Senior Software Engineer, Red Hat

10 Gb Ethernet - Overview

- This presentation is about 10Gb Ethernet performance, tuning and functionality in RHEL5.X
  - RHEL4.X also contain support for 10GbE cards
    - Needed to limit our focus based on resources for summit
- This talk will cover:
  - Platform characteristics to look for
  - Tools that I find useful to debug or tune
  - Tuning steps
  - Performance of 10GbE networking in:
    - A virtualized environment
    - MRG – Messaging, Realtime, Grid

Some Quick Disclaimers

- Test results based on two different platforms
  - Intel, AMD
- Cards supplied by three different vendors
  - Chelsio, Intel, Neterion
- Red Hat supports all of devices used for this presentation
  - We do not recommend one over the other
- Testing based on “performance mode”
  - Maximize a particular thing at the expense of other things
  - Not recommended for production
- Don't assume settings shown will work for you without some tweaks

Take Aways

- Hopefully, you will be able to leave this talk with:
  - An understanding of the tools available to help you evaluate your network performance
  - An understanding of 10GbE performance under RHEL5
- Use this talk as suggestions of things to try
  - My testing based on local network – wide area network will be different
  - Do not assume all setting will work for you without some tweaks

Take Aways - continued

- Read the vendors Release Notes, tuning guides, etc
  - Visit their website
  - Install and read the source
- `/usr/src/redhat/BUILD/kernel-2.6.18/linux-2.6.18.x86_64/Documentation/networking`
- Use the Internet as starting place for the knobs to turn
  - Do not assume every setting on the Internet will work for you, often times different websites conflict with their advice
    - Internet search “linux tcp_window_scaling performance”
    - some sites say to set it to 0, other sites set it to 1
      - Let's run a test to see what works best in *my* testbed....

A Quick Example

Internet search for "linux tcp_window_scaling performance" will show some sites say to set it to 0 others say set it to 1

```
[root@perf12 np2.4]# sysctl -w net.ipv4.tcp_window_scaling=0
[root@perf12 np2.4]# ./netperf -P1 -l 30 -H 192.168.10.100
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec

 87380  16384  16384    30.00    2106.40
```

```
[root@perf12 np2.4]# sysctl -w net.ipv4.tcp_window_scaling=1
[root@perf12 np2.4]# ./netperf -P1 -l 30 -H 192.168.10.100
Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec

 87380  16384  16384    30.01    5054.68
```

Platform Features

- Multiple processors
- Fast memory
- All my testing has been done on PCIe
  - PCI-X not really fast enough for full-duplex
  - Look for width of 8 lanes (8x)
    - PCIe is typically 250 Gb/sec per lane
- Look for support of MSI-X interrupts
  - Server
  - OS (RHEL does this :)
  - Driver

Tools

- Monitor / debug tools
  - mpstat – reveals per cpu stats, Hard/Soft Interrupt usage
  - vmstat – vm page info, context switch, total ints/s, cpu
  - netstat – per nic status, errors, statistics at driver level
  - lspci – list the devices on pci, indepth driver flags
  - oprofile – system level profiling, kernel/driver code
  - modinfo – list information about drivers, version, options
  - sar – collect, report, save system activity information
  - Many others available- iptraf, wireshark, etc
- Sample use for some of these embedded in talk

Tools (cont)

- Tuning tools
  - ethtool – View and change Ethernet card settings
  - sysctl – View and set */proc/sys* settings
  - ifconfig – View and set ethX variables
  - setpci – View and set pci bus params for device
  - netperf – Can run a bunch of different network tests
  - /proc – OS info, place for changing device tunables

ethtool

- Works mostly at the HW level
  - ethtool -S – provides HW level stats
    - Counters since boot time, create scripts to calculate diffs
  - ethtool -c - Interrupt coalescing
  - ethtool -g - provides ring buffer information
  - ethtool -k - provides hw assist information
  - ethtool -i - provides the driver information

ethtool -c - interrupt coalesce

```
[root@perf10 ~]# ethtool -c eth2
Coalesce parameters for eth2:
Adaptive RX: off  TX: off
stats-block-usecs: 0
sample-interval: 0
pkt-rate-low: 0
pkt-rate-high: 0

rx-usecs: 5
rx-frames: 0
rx-usecs-irq: 0
rx-frames-irq: 0

tx-usecs: 0
tx-frames: 0
tx-usecs-irq: 0
tx-frames-irq: 0

<truncated>
```

ethtool -g - HW Ring Buffers

```
[root@perf10 ~]# ethtool -g eth2
Ring parameters for eth2:
Pre-set maximums:
RX:             16384
RX Mini:        0
RX Jumbo:       16384
TX:             16384
Current hardware settings:
RX:             1024
RX Mini:        1024
RX Jumbo:       512
TX:             1024
```

Typically these numbers correspond the number of buffers, not the size of the buffer

With some NICs creating more buffers decreases the size of each buffer which could add overhead

ethtool -k - HW Offload Settings

```
[root@perf10 ~]# ethtool -k eth2
Offload parameters for eth2:
Cannot get device udp large send offload settings: Operation
  not supported
rx-checksumming: on
tx-checksumming: on
scatter-gather: on
tcp segmentation offload: on
udp fragmentation offload: off
generic segmentation offload: off
```

These provide the ability to offload the CPU for calculating the checksums, etc.

ethtool -i - driver information

```
[root@perf10 ~]# ethtool -i eth2
driver: cxgb3
version: 1.0-ko
firmware-version: T 5.0.0 TP 1.1.0
bus-info: 0000:06:00.0

[root@perf10 ~]# ethtool -i eth3
driver: ixgbe
version: 1.1.18
<truncated>

[root@dhcp47-154 ~]# ethtool -i eth2
driver: Neterion (ed. note s2io)
version: 2.0.25.1
<truncated>
```

sysctl

- sysctl is a mechanism to view and control the entries under the /proc/sys tree
- sysctl -a - lists all variables
- sysctl -q - queries a variable
- sysctl -w - writes a variable
  - When setting values, spaces are not allowed
    - `sysctl -w net.ipv4.conf.lo.arp_filter=0`
- Setting a variable via sysctl on the command line is **not persistent** The change is only valid until the next reboot
  - Write entries into the /etc/sysctl.conf file to have them applied at boot time

Some Important settings for sysctl

- Already showed tcp_window_scaling issue
- By default, Linux networking not tuned for max performance, more for reliability
  - Buffers are especially not tuned for local 10GbE traffic
  - Remember that Linux “autotunes” buffers for connections
  - Don't forget UDP !
- Try via command line
  - When you are happy with the results, add to /etc/sysctl.conf
- Look at documentation in `/usr/src`
  - /usr/src/redhat/BUILD/kernel-2.6.18/linux-2.6.18.x86_64/Documentation/networking

Some Important settings for sysctl

- net.ipv4.tcp_window_scaling - toggles window scaling
- Misc TCP protocol
  - net.ipv4.tcp_timestamps - toggles TCP timestamp support
  - net.ipv4.tcp_sack - toggles SACK (Selective ACK) support
- TCP Memory Allocations - min/pressure/max
  - net.ipv4.tcp_rmem - TCP read buffer - in bytes
    - overriden by core.rmem_max
  - net.ipv4.tcp_wmem - TCP write buffer - in bytes
    - overridden by core/wmem_max
  - net.ipv4.tcp_mem - TCP buffer space
    - measured in pages, not bytes !

Some Important settings for sysctl

- CORE memory settings
  - net.core.rmem_max - max size of rx socket buffer
  - net.core.wmem_max -max size of tx socket buffer
  - net.core.rmem_default - default rx size of socket buffer
  - net.core.wmem_default - default tx size of socket buffer
  - net.core.optmem_max - maximum amount of option memory buffers
- net.core.netdev_max_backlog how many unprocessed rx packets before kernel starts to drop them
- These settings also impact UDP

netperf

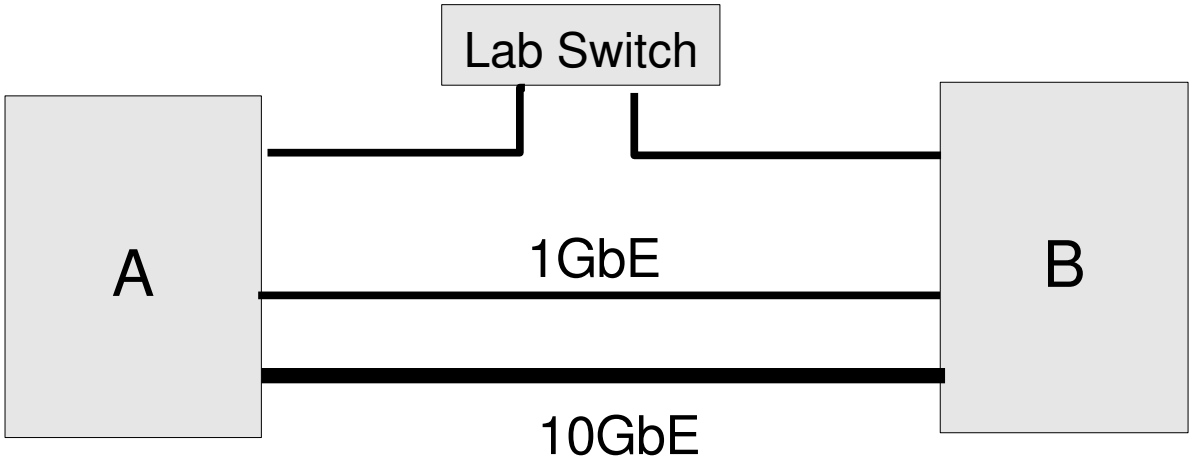
- http://netperf.org
- Feature Rich
  - Read documentation
  - Default test is TCP_STREAM – uses *send()* call
  - TCP_SENDFILE – uses *sendfile()* call – much less copying
  - TCP_RR – Request / Response tests
  - UDP_STREAM
  - Many others

Know what you are testing

- Linux has several automatic features that may cause unanticipated side effects
  - Message delivery - Linux does its best to get message from A to B
    - Packet may get from A to B via different path than you think
    - Check arp_filter settings - **`sysctl -a | grep arp_filter`**
  - Automatic buffer sizing
    - Be explicit if it matters to you

Control your network route :

- Check arp_filter settings with sysctl
  - sysctl -a | grep arp_filter
    - A setting of 0 says uses any path
  - If more than one path between machines, set arp_filter=1
  - Look for increasing interrupt counts in /proc/interrupt or increasing counters via ifconfig or netstat



Know what you are testing - Hardware

- Did the PCIe bus get negotiated correctly?
  - Use lspci
- Did the interrupts come up as expected
  - MSI-X can make a big difference
    - On some cards its not on by default
- Several vendors have information on changing the default PCI-E settings via setpci
  - Read the Release Notes / README !

lspci – validate your slot setting for each NIC

lspci -v -v -s 09:00.0

```
09:00.0 Ethernet controller: 10GbE Single Port Protocol Engine Ethernet Adapter
                          < truncated >
        Capabilities: [58] Express Endpoint IRQ 0
                Device: Supported: MaxPayload 4096 bytes, PhantFunc 0, ExtTag+
                Device: Latency L0s <64ns, L1 <1us
                Device: AtnBtn- AtnInd- PwrInd-
                Device: Errors: Correctable- Non-Fatal- Fatal- Unsupported-
                Device: RlxdOrd+ ExtTag- PhantFunc- AuxPwr- NoSnoop+
                Device: MaxPayload 128 bytes, MaxReadReq 512 bytes
                Link: Supported Speed 2.5Gb/s, Width x8, ASPM L0s L1, Port 0
                Link: Latency L0s unlimited, L1 unlimited
                Link: ASPM Disabled RCB 64 bytes CommClk- ExtSynch-
                Link: Speed 2.5Gb/s, Width x4
                Vector table: BAR=4 offset=00000000
```

Some General System Tuning Guidelines

- To maximize network throughput lets
  - Disable irqbalance
    - `service irqbalance stop`
    - `chkconfig irqbalance off`
  - Disable cpuspeed
    - default gov=ondemand, set governer to performance
  - Use affinity to maximize what WE want
    - Process affinity
      - Use `taskset or MRG's "Tuna"`
    - Interrupt affinity
      - `grep eth2 /proc/interrupts`
      - `echo 80 > /proc/irq/177/smp_affinity`

Performance Tuning Outline

- IRQ Affinity / Processor Affinity - No magic formula
  - experiment to get the best results
  - Interrupt coalescing
- *** My** * experience is that chip architectures play a big role
- Try to match TX and RX on same socket / data caches
- sysctl.conf
  - Increase/decrease memory parameter for **network**
- Driver Setting
  - NAPI – if driver supports
  - HW ring buffers
  - TSO, UFO, GSO

Actual Tuning Example

- You just got those new 10GbE cards that **you** told the CIO would greatly improve performance
- You plug them in and run a quick netperf to verify your choice

New Boards, first Run

```
# ./netperf -P1 -l 60 -H 192.168.10.10

Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec

 87380  16384  16384    60.00    5012.24
```

Hmm, about 5 Gb/sec, half of what the CIO is expecting

Lets see where the bottleneck is :

New Boards, first run mpstat -P ALL 5

Transmit

CPU	%sys	%iowait	%irq	%soft	%steal	%idle	intr/s
all	2.17	0.00	0.35	1.17	0.00	96.23	12182.00
0	17.40	0.00	2.80	9.20	0.00	70.00	12182.00
1	0.00	0.00	0.00	0.00	0.00	100.00	0.00
2	0.00	0.00	0.00	0.00	0.00	100.00	0.00
3	0.00	0.00	0.00	0.00	0.00	100.00	0.00
4	0.00	0.00	0.00	0.00	0.00	100.00	0.00
5	0.00	0.00	0.00	0.00	0.00	100.00	0.00
6	0.00	0.00	0.00	0.00	0.00	100.00	0.00
7	0.00	0.00	0.00	0.00	0.00	100.00	0.00

Receive

CPU	%sys	%iowait	%irq	%soft	%steal	%idle	intr/s
all	4.86	0.00	0.07	7.56	0.00	87.49	10680.90
0	**38.90**	**0.00**	**0.60**	**60.40**	**0.00**	**0.00**	**10680.90**
1	0.00	0.00	0.00	0.00	0.00	100.00	0.00
2	0.00	0.00	0.00	0.00	0.00	100.00	0.00
3	0.00	0.00	0.00	0.00	0.00	100.00	0.00
4	0.00	0.00	0.00	0.00	0.00	100.00	0.00
5	0.00	0.00	0.00	0.00	0.00	100.00	0.00
6	0.00	0.00	0.00	0.00	0.00	100.00	0.00
7	0.00	0.00	0.00	0.00	0.00	100.00	0.00

Tuning – Identify the Bottlenecks

- Run "`mpstat -P ALL 5`" while running netperf
- Review of output shows core0 on Receive side is pegged
- So lets try to set some IRQ affinity
  - `grep` for the NIC in `/proc/interrupts`
  - Echo the desired value into `/proc/irq/XXX/smp_affinity`
    - Does NOT persist across reboots

Setting IRQ Affinity

- CPU cores designated by bitmap
- `cat /proc/cpuinfo` to determine how the BIOS presented the CPUs to the system
  - Some go Socket0, core0, socket1, core0
  - Others go Socket0, core0, socket0, core1
- Understand the layout of L2 cache in relationship to the cores
- Remember these values do not persistent across reboots!
- Set IRQ affinity
  - `echo 80 > /proc/irq/192/smp_affinity`
  - Use “TUNA”

Know Your CPU core layout

```
# cat /proc/cpuinfo
```

```
processor       : 0
physical id     : 0
core id         : 0

processor       : 2
physical id     : 0
core id         : 1

processor       : 4
physical id     : 0
core id         : 2

processor       : 6
physical id     : 0
core id         : 3
```

Socket 0

```
processor       : 1
physical id     : 1
core id         : 0

processor       : 3
physical id     : 1
core id         : 1

processor       : 5
physical id     : 1
core id         : 2

processor       : 7
physical id     : 1
core id         : 3
```

Socket1

Setting IRQ Affinity

Now lets move the interrupts, remember your core mapping is important

Note that the separate irq for TX and RX

```
Transmit
# grep eth2 /proc/interrupts
        CPU0   CPU1   CPU2   CPU3   CPU4   CPU5   CPU6   CPU7
 74:  360345      0      0      0      0      0      0      0   PCI-MSI-X  eth2-tx0
 82:  647960      0      0      0      0      0      0      0   PCI-MSI-X  eth2-rx0
 90:       0      0      0      0      0      0      0      0   PCI-MSI-X  eth2-lsc
# echo 40 > /proc/irq/74/smp_affinity
# echo 80 > /proc/irq/82/smp_affinity

Receive
# grep eth2 /proc/interrupts
        CPU0   CPU1   CPU2   CPU3   CPU4   CPU5   CPU6   CPU7
194:    6477      0      0      0      0      0      0      0 PCI-MSI-X eth2
202: 5795405      0      0      0      0      0      0      0 PCI-MSI-X eth2(queue0)
# echo 40 > /proc/irq/194/smp_affinity
# echo 80 > /proc/irq/202/smp_affinity
```

Tuning – Run2, IRA Affinity

```
# ./netperf -P1 -l 30 -H 192.168.10.10

Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec

 87380  16384  16384    30.00    5149.89
```

OK, so throughput is up slightly, let's look for bottlenecks

Run 2 – mpstat -P ALL 5 outputs

Transmit

CPU	%sys	%iowait	%irq	%soft	%steal	%idle	intr/s
all	2.35	0.00	0.43	1.43	0.00	95.75	12387.80
0	0.00	0.00	0.00	0.00	0.00	100.00	1018.00
1	0.00	0.00	0.00	0.00	0.00	100.00	0.00
2	0.00	0.00	0.00	0.00	0.00	100.00	0.00
3	0.00	0.00	0.00	0.00	0.00	100.00	0.00
4	0.00	0.00	0.00	0.00	0.00	100.00	0.00
5	0.00	0.00	2.80	2.00	0.00	95.20	4003.80
6	18.60	0.00	0.60	9.40	0.00	70.80	7366.40
7	0.00	0.00	0.00	0.00	0.00	100.00	0.00

Receive

CPU	%sys	%iowait	%irq	%soft	%steal	%idle	intr/s
all	4.67	0.00	0.07	7.75	0.00	87.49	10989.79
0	0.00	0.00	0.00	0.00	0.00	100.00	1018.52
1	0.00	0.00	0.00	0.00	0.00	100.00	0.00
2	0.00	0.00	0.00	0.00	0.00	100.00	0.00
3	0.00	0.00	0.00	0.00	0.00	100.00	0.00
4	0.00	0.00	0.00	0.00	0.00	100.00	0.00
5	0.00	0.00	0.00	0.00	0.00	100.00	0.00
6	0.00	0.00	0.00	0.00	0.00	100.00	0.00
7	**37.36**	**0.00**	**0.60**	**61.94**	**0.00**	**0.00**	**9971.17**

Run 2 – review data – next steps

- We moved interrupts and reran the test,
  - saw a very slight improvement in throughput,
    - not really expecting much...yet
- Looking at the mpstat output we can see that we are still bottlenecked on the receive side.
- Let’s now try to add some process affinity to the mix
  - Remember core mappings
  - Try to get netperf to run on a “sister” core to the interrupts
  - Typically use taskset or TUNA for this, can also handle programatically
    - netperf also has a built in mechanism

Run 3 – Add Process Affinity

```
# ./netperf -P1 -l 30 -H 192.168.10.10 -T 5,5

Recv   Send    Send
Socket Socket  Message  Elapsed
Size   Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec

 87380  16384  16384    30.00    4927.19
```

Throughput is down slightly , let's see where the bottleneck is now

Run3 - Bottlenecks

Transmit

CPU	%sys	%iowait	%irq	%soft	%steal	%idle	intr/s
all	4.35	0.00	0.35	16.00	0.00	79.25	11272.55
0	0.00	0.00	0.00	0.00	0.00	100.00	1020.44
1	0.00	0.00	0.00	0.00	0.00	100.00	0.00
2	0.00	0.00	0.00	0.00	0.00	100.00	0.00
3	0.00	0.00	0.00	0.00	0.00	100.00	0.00
4	0.00	0.00	0.00	0.00	0.00	100.00	0.00
5	**34.73**	**0.00**	**2.59**	**62.08**	**0.00**	**0.00**	**4011.82**
6	0.00	0.00	0.40	65.60	0.00	34.00	6240.28
7	0.00	0.00	0.00	0.00	0.00	100.00	0.00

Receive

CPU	%sys	%iowait	%irq	%soft	%steal	%idle	intr/s
all	3.85	0.00	0.10	17.62	0.00	78.39	17679.00
0	0.00	0.00	0.00	0.00	0.00	100.00	1016.80
1	0.00	0.00	0.00	0.00	0.00	100.00	0.00
2	0.00	0.00	0.00	0.00	0.00	100.00	0.00
3	0.00	0.00	0.00	0.00	0.00	100.00	0.00
4	0.00	0.00	0.00	0.00	0.00	100.00	0.00
5	**30.90**	**0.00**	**0.00**	**63.70**	**0.00**	**5.10**	**0.00**
6	0.00	0.00	0.00	0.00	0.00	100.00	0.00
7	**0.00**	**0.00**	**0.80**	**77.30**	**0.00**	**21.90**	**16661.90**

Run3, Analysis + next steps

- By adding process affinity things have changed
  - The bottleneck is now on the transmit side
  - core5 on TX is 100%, core5 on RX side is handling the load
- Try moving process affinities around (already done)
- Change the code (if you can)
  - Default netperf method uses *send()* which copies data around
  - Try TCP_SENDFILE which use the *sendfile()* system call
- Try bigger MTU
  - Currently at 1500

Run 4 – Change send() to sendfile()

```
#./netperf -P1 -l 30 -H 192.168.10.10 -T 5,5 -t TCP_SENDFILE -F
   /data.file

Recv    Send    Send
Socket Socket  Message  Elapsed
Size    Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec

 87380  16384  16384    30.00    6689.77
```

Run 4 – mpstat output – sendfile option

Transmit

CPU	%sys	%iowait	%irq	%soft	%steal	%idle	intr/s
all	1.55	0.00	0.38	7.30	0.00	90.70	12645.00
0	0.00	0.00	0.00	0.00	0.00	100.00	1018.20
1	0.00	0.00	0.00	0.00	0.00	100.00	0.00
2	0.00	0.00	0.00	0.00	0.00	100.00	0.00
3	0.00	0.00	0.00	0.00	0.00	100.00	0.00
4	0.00	0.00	0.00	0.00	0.00	100.00	0.00
5	**12.38**	**0.00**	**2.20**	**14.17**	**0.00**	**70.66**	**3973.40**
6	0.00	0.00	0.80	44.20	0.00	55.00	7653.20
7	0.00	0.00	0.00	0.00	0.00	100.00	0.00

Receive

CPU	%sys	%iowait	%irq	%soft	%steal	%idle	intr/s
all	5.73	0.31	0.04	18.49	0.00	75.00	6050.75
0	0.20	2.50	0.10	0.10	0.00	95.20	1051.65
1	0.00	0.00	0.00	0.00	0.00	100.00	0.00
2	0.10	0.00	0.00	0.00	0.00	99.90	0.00
3	0.30	0.00	0.00	0.00	0.00	98.20	0.00
4	0.00	0.00	0.00	0.00	0.00	100.00	0.00
5	45.25	0.00	0.00	52.75	0.00	1.90	0.00
6	0.00	0.00	0.00	0.00	0.00	100.00	2.70
7	0.00	0.00	0.40	95.01	0.00	4.59	4996.60

Tuning - Identifying the Bottlenecks

- Wow, our core with the netperf process on TX went from 0% idle to 70% idle by switching the system call
  - Overall the system reclaimed 10%
- Nice jump in throughput, but we are still not near 10Gb
- Let's try a larger MTU
  - `ifconfig eth2 mtu 9000 up`
    - Note this causes a temporary drop in the connection

Run 5 – Kick up MTU = 9000

```
#./netperf -P1 -l 30 -H 192.168.10.10 -T 5,5 -t TCP_SENDFILE -F
   /data.file

Recv    Send    Send
Socket Socket  Message  Elapsed
Size    Size    Size     Time     Throughput
bytes  bytes   bytes    secs.    10^6bits/sec

  87380  16384  16384    30.00    9888.66
```

OK, now we can show this to the CIO

Run 5 – Kick up MTU = 9000

TX

CPU	%sys	%iowait	%irq	%soft	%steal	%idle	intr/s
all	1.40	0.00	0.32	4.27	0.00	93.90	13025.80
0	0.00	0.00	0.00	0.00	0.00	100.00	1015.00
1	0.00	0.00	0.00	0.00	0.00	100.00	0.00
2	0.00	0.00	0.00	0.00	0.00	100.00	0.00
3	0.00	0.00	0.00	0.00	0.00	100.00	0.00
4	0.00	0.00	0.00	0.00	0.00	100.00	0.00
5	11.00	0.00	2.20	7.40	0.00	78.80	4003.80
6	0.00	0.00	0.40	26.60	0.00	73.00	8007.20
7	0.00	0.00	0.00	0.00	0.00	100.00	0.00

RX

CPU	%sys	%iowait	%irq	%soft	%steal	%idle	intr/s
all	6.63	0.00	0.39	5.40	0.00	87.10	69932.10
0	0.00	0.00	0.00	0.00	0.00	100.00	1017.80
1	0.00	0.00	0.00	0.00	0.00	100.00	0.00
2	0.00	0.00	0.00	0.00	0.00	100.00	0.00
3	0.00	0.00	0.00	0.00	0.00	100.00	0.00
4	0.00	0.00	0.00	0.00	0.00	100.00	0.00
5	53.00	0.00	0.00	20.20	0.00	22.90	0.00
6	0.00	0.00	0.00	0.00	0.00	100.00	0.00
7	0.00	0.00	3.10	23.02	0.00	73.87	68914.20

Features - Multi-queue

- RHEL5 has support for several vendors RX multi-queue
  - Typically enabled via script or module parameters
- Still no TX multi-queue (that I know of at least)
- RX Multi-queue big gainer when there are multiple applications using the network
    - Use affinity for queues and match queue to task (taskset)
  - Potential advantage with single version of netperf if you have slower CPUs / memory

Single RX Queue – Multiple netperf

```
  1016.95     192.168.10.37
   819.41     192.168.10.12
   898.93     192.168.10.17
   961.87     192.168.10.16
------------
  3696
```

CPU	%sys	%iowait	%irq	%soft	%steal	%idle	intr/s
all	3.90	0.00	0.00	23.63	0.00	72.44	1054.80
0	**0.00**	**0.00**	**0.00**	**100.00**	**0.00**	**0.00**	**1054.80**
1	0.20	0.00	0.00	0.20	0.00	99.60	0.00
2	8.20	0.00	0.00	19.00	0.00	72.80	0.00
3	8.22	0.00	0.00	24.05	0.00	67.74	0.00
4	0.00	0.00	0.00	0.00	0.00	100.00	0.00
5	7.40	0.00	0.00	24.80	0.00	67.80	0.00
6	7.21	0.00	0.00	21.24	0.00	71.54	0.00
7	0.00	0.00	0.00	0.00	0.00	100.00	0.00

Multiple RX Queues, multiple netperfs

```
  1382.25    192.168.10.37
  2127.18    192.168.10.17
  1726.71    192.168.10.16
  1986.31    192.168.10.12
  -------
  7171
```

```
CPU    %sys %iowait    %irq   %soft  %steal   %idle    intr/s
all    6.55    0.00    0.18   42.84    0.00   50.44  28648.00
  0    2.40    0.00    0.00    6.60    0.00   91.00   1015.40
  1   11.45    0.00    0.40   83.13    0.00    5.02   9825.00
  2    2.00    0.00    0.20   97.80    0.00    0.00   2851.80
  3    0.00    0.00    0.00    0.00    0.00  100.00      0.00
  4   10.60    0.00    0.00   36.20    0.00   53.20      0.00
  5    0.00    0.00    0.00    0.00    0.00  100.00      0.00
  6   11.22    0.00    0.00   35.87    0.00   52.91      0.00
  7   14.77    0.00    0.80   83.03    0.00    1.20  14955.80
```

Interrupt distribution w/ MultiQueue

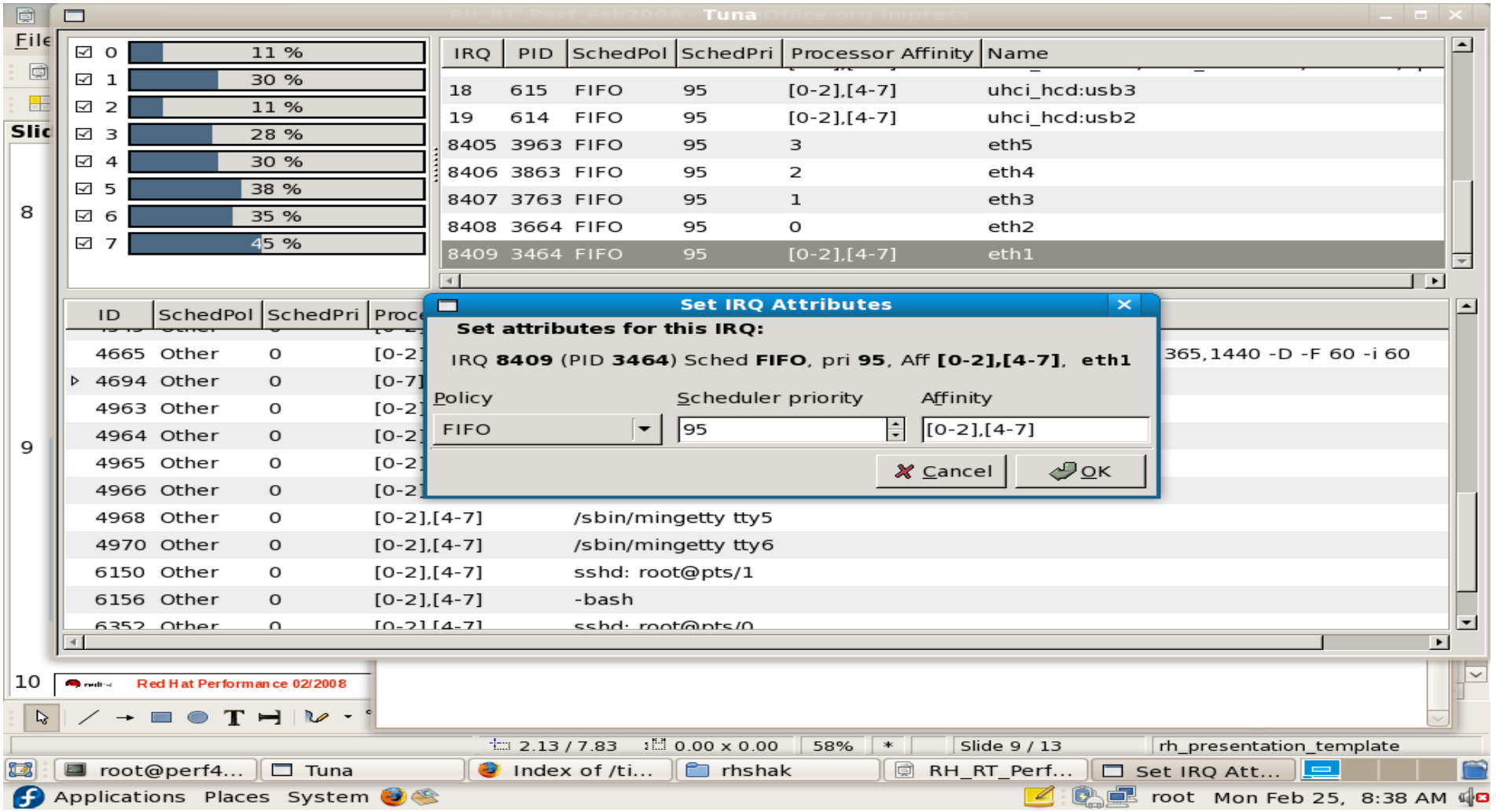
```
[]# grep eth2 /proc/interrupts

  CPU0       CPU1       CPU2       CPU3       CPU4       CPU5       CPU6       CPU7
130: 5          0          0          0          0          0          0          0   eth2
138:24 11032798          0          0          0          0          0          0   eth2 q0
146: 1          0    1821803          0          0          0          0          0   eth2 q1
154: 1          0          0    5242518          0          0          0          0   eth2 q2
162: 1          0          0          0    1849812          0          0          0   eth2 q3
170: 1          0          0          0          0    7301950          0          0   eth2 q4
178: 1          0          0          0          0          0    8426940          0   eth2 q5
186: 1          0          0          0          0          0          0    1809018   eth2 q6
```

MRG and 10GbE

- A different ballgame
- Latency often trumps throughput
  - But most people want both
- Messaging tends to leverage UDP and small packets
  - Predictability is important

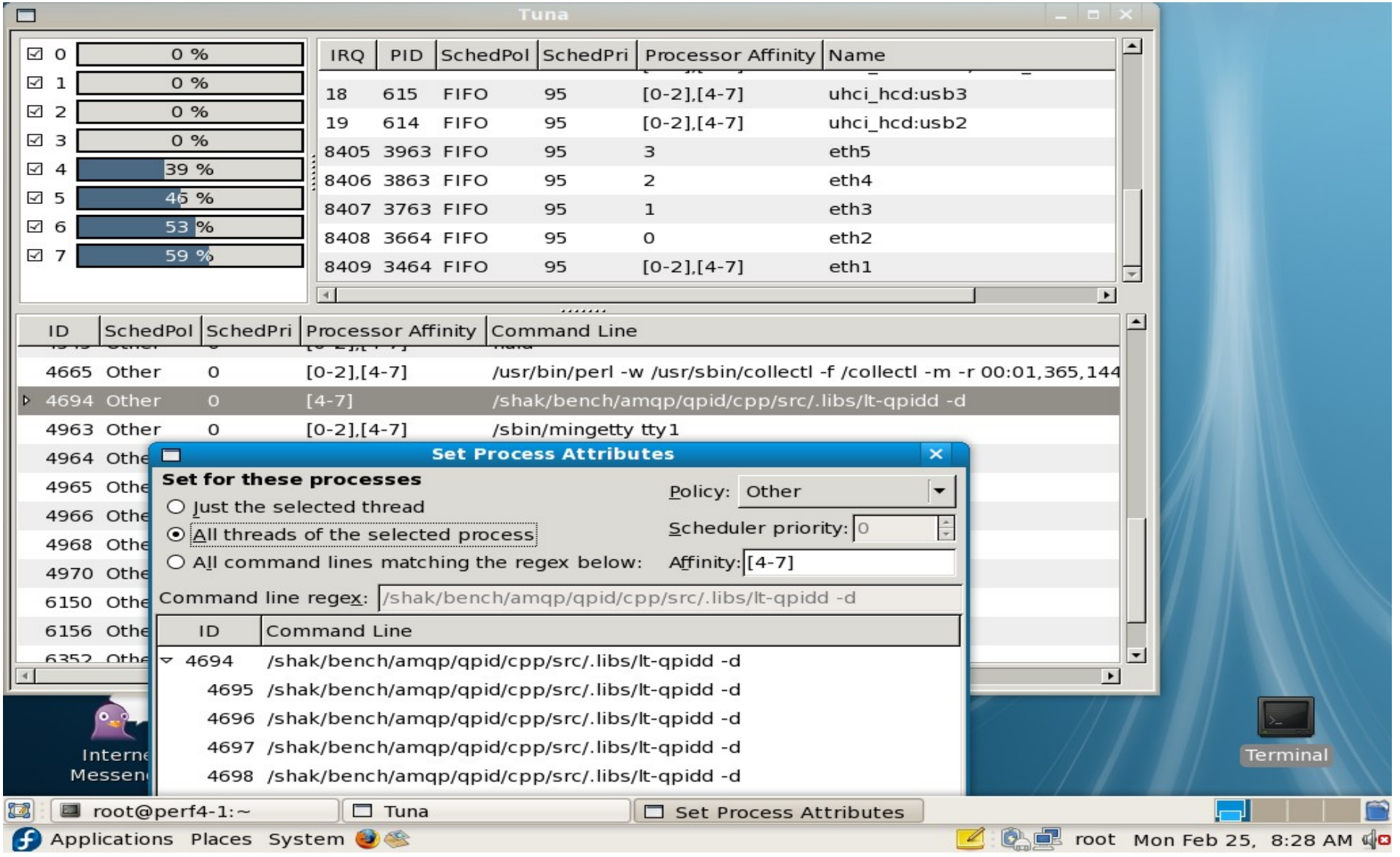
New Tuning Tools w/ RH MRG

MRG Tuning – using the TUNA – dynamically control
- Device IRQ properties
  - CPU affinity / parent and threads
  - Scheduling policy



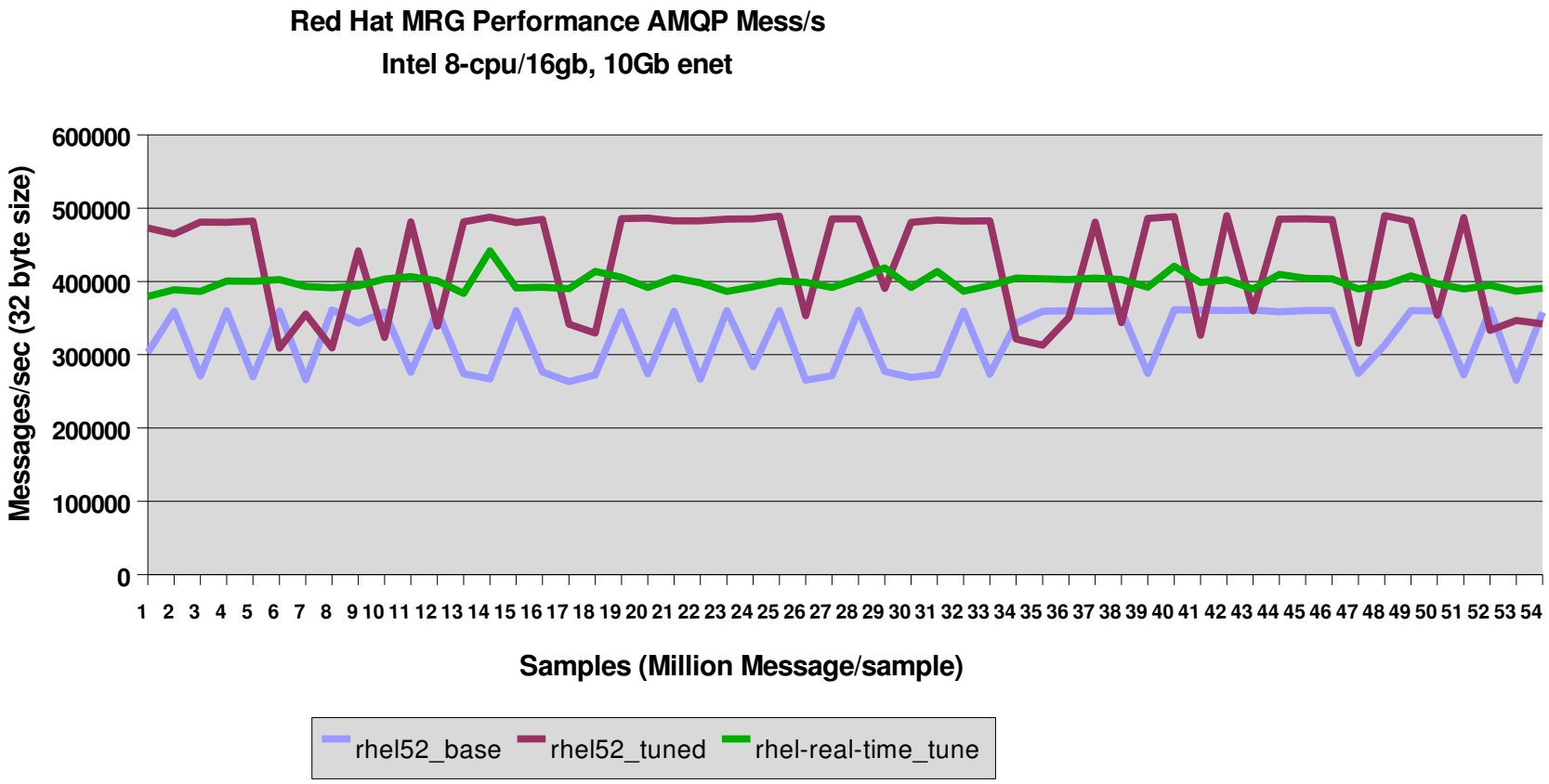
New Tuning Tools w/ RH MRG

MRG Tuning – using the TUNA – dynamically control
- Process affinity / parent and threads
- Scheduling policy



Tuning Network Apps Messages/sec

10 Gbit Nics Stoakley 2.67 to Bensley 3.0 Ghz
Tuning enet gains +25% in Ave Latency,
RT kernel reduced peak latency but smoother – how much?



Latency

- The interrupt coalescing settings are vital
  - ethtool -c eth3 to read , -C to set
    - Rx-usecs tells how often to service interrupt
  - NAPI may help as you can handle multiple interrupts
- Also look at using TCP_NODELAY options
  - May help with latency but hurt throughput
  - “Nagles Algorithm” tries to fill packets in order to avoid overhead of sending

Lower RX Latency with ethtool -C

```
# ethtool -c eth6
Coalesce parameters for eth6:
<truncate>
rx-usecs: 125
rx-frames: 0
rx-usecs-irq: 0
rx-frames-irq: 0

# ./netperf -H 192.168.10.12 -t TCP_RR

Local /Remote
Socket Size   Request  Resp.   Elapsed  Trans.
Send   Recv   Size     Size    Time     Rate
bytes  Bytes  bytes    bytes   secs.    per sec

16384  87380  1        1       10.00    8000.27
```

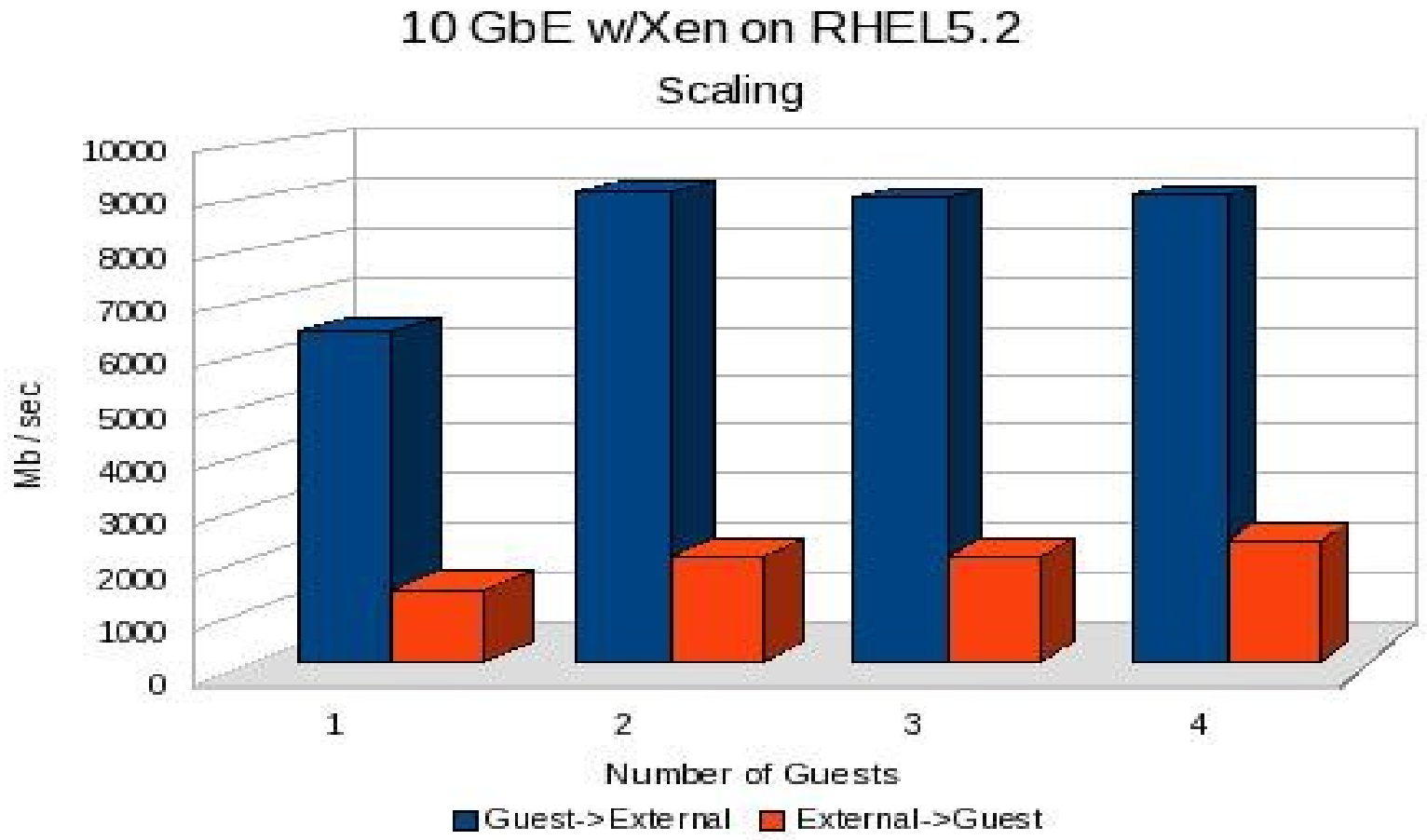
Lower rx-usecs on the receiver and rerun

```
# ethtool -C eth6 rx-usecs 100

# ./netperf -H 192.168.10.12 -t TCP_RR

16384  87380  1        1       10.00    10009.83
```

10GbE in a Virtual World

- Single guest w/ 1Gb interface – Good
- Single guest w/ 10 Gb interface - Better than 1Gb
  - Several copies needed for security
    - cpu/memory BW limits 10Gbit performance
    - Better than 1Gb network but not wire speed.
    - Same speed as using a dummy network
- Use RHEL5.2 nic limit

10 GbE Scaling w/Xen on RHEL5.2



10GbE in a Virtual World - cont

- Going forward:
  - PCI_passthru looks promising for performance
    - Guest has direct control over NIC
  - Network throughput for fully-virt/HVM guests will be limited to 100Mb/s. It can be improved by installing the para-virt drivers for fully-virt/HVM RHEL/Windows guests
  - kvm has virtio work going on to improve performance

Wrap up

- There are lots of knobs to use, the trick is finding them and learning how to use them
- Learn to use the tools to help investigate issues
- Full-Duplex (20Gb/sec) is possible under RHEL5
- Questions ?

Credits

- The following helped me along the way in preparing this presentation, to them a special thank you
  - Andy Gospo
  - Don Dutile
  - D John Shakshober
  - JanMark Holzer