Performance Tuning of Gigabit Network Interfaces

Robert Geist

James Martin

James Westall

Venkat Yalla

rmg,jmarty,westall,vyalla@cs.clemson.edu

Outline

- The problem
- The network testbed
- The workload
- The results
- The model
- Conclusions

The Problem

- A standard DIX Ethernet frame consists of
  - Physical and link layer headers: 22 bytes
  - Network layer PDU: 1500 bytes
  - Link and physical layer trailers: 16 bytes
  - Or a total of 12304 bits

The Problem

- A standard DIX Ethernet frame consists of
  - Physical and link layer headers: 22 bytes
  - Network layer PDU: 1500 bytes
  - Link and physical layer trailers: 16 bytes
  - Or a total of 12304 bits

- At 1 Gigabit per second
  - The arrival time between frames is 12.3 μsec
  - and the maximum arrival rate is $81,324 frames/sec$

The Problem

- A standard DIX Ethernet frame consists of
  - Physical and link layer headers: 22 bytes
  - Network layer PDU: 1500 bytes
  - Link and physical layer trailers: 16 bytes
  - Or a total of 12304 bits

- At 1 Gigabit per second
  - The arrival time between frames is 12.3 μsec
  - and the maximum arrival rate is $81,324 frames/sec$

- At this frame rate the CPU can become a bottleneck leading to
  - Degraded system performance,
  - Poor network throughput, or in the worst case
  - Receive livelock

Sources of CPU Overhead

- Per interrupt overhead
  - A context switch occurs when the NIC generates an interrupt
  - Another context switch occurs at the end of interrupt service.

Sources of CPU Overhead

- Per interrupt overhead
  - A context switch occurs when the NIC generates an interrupt
  - Another context switch occurs at the end of interrupt service.
- Per packet overhead
  - Processing of link, network and transport headers
  - Kernel buffer and buffer queue management

Sources of CPU Overhead

- Per interrupt overhead
  - A context switch occurs when the NIC generates an interrupt
  - Another context switch occurs at the end of interrupt service.
- Per packet overhead
  - Processing of link, network and transport headers
  - Kernel buffer and buffer queue management
- Per byte overhead
  - Copying data from kernel to user space buffers.
  - Software checksumming (if required)

Overhead mitigation

- Coalesce interrupts
  - When a frame arrival completes, the NIC delays before raising IRQ
  - If a new arrival commences during the delay, the NIC defers the IRQ until frame completes
  - Repeat until some absolute time limit is reached or receive buffers run low.

Overhead mitigation

- Coalesce interrupts
  - When a frame arrival completes, the NIC delays before raising IRQ
  - If a new arrival commences during the delay, the NIC defers the IRQ until frame completes
  - Repeat until some absolute time limit is reached or receive buffers run low.
- Use larger frame sizes
  - Packet processing costs are amortized over larger payloads
  - Longer frame times imply lower interrupt rates as well.

Overhead mitigation

- Coalesce interrupts
  - When a frame arrival completes, the NIC delays before raising IRQ
  - If a new arrival commences during the delay, the NIC defers the IRQ until frame completes
  - Repeat until some absolute time limit is reached or receive buffers run low.
- Use larger frame sizes
  - Packet processing costs are amortized over larger payloads
  - Longer frame times imply lower interrupt rates as well.
- Use zero copy protocols in which NIC transfers data directly to user space buffers with DMA

Complicating factors

- Coalescing of interrupts
  - Not all NIC's support interrupt coalescing
  - Even if the NIC does the device driver may not
  - Even if the device driver does support it configuring it may be an adventure.
  - But at least coalescing does*not raise* the interoperability issues of large frames

Complicating factors

- Coalescing of interrupts
  - Not all NIC's support interrupt coalescing
  - Even if the NIC does the device driver may not
  - Even if the device driver does support it configuring it may be an adventure.
  - But at least coalescing does*not raise* the interoperability issues of large frames
- Use of large frame sizes raises interoperability issues
  - Maximum frame size supported varies by NIC and switch
  - Some NICs don't support large frames at all
  - Some switches will not pass large frames at all

Complicating factors

- Coalescing of interrupts
  - Not all NIC's support interrupt coalescing
  - Even if the NIC does the device driver may not
  - Even if the device driver does support it configuring it may be an adventure.
  - But at least coalescing does*not raise* the interoperability issues of large frames
- Use of large frame sizes raises interoperability issues
  - Maximum frame size supported varies by NIC and switch
  - Some NICs don't support large frames at all
  - Some switches will not pass large frames at all
- Use of zero copy protocols
  - Requires replacement of complete protocol stack

Testbed

- A 265 node Beowulf type cluster
- Extreme Black Diamond 6804 Switch provides two VLANs
  - Management VLAN: (rlogin, NFS, etc)
  - Graphics VLAN: (used in the tests reported here)

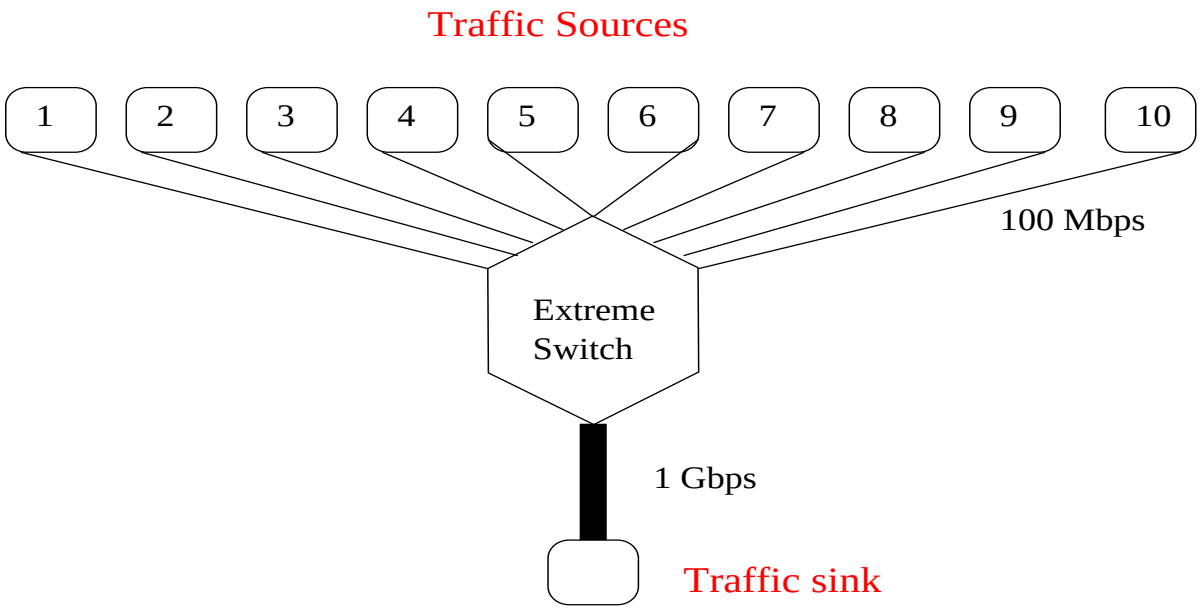
Testbed

- A 265 node Beowulf type cluster
- Extreme Black Diamond 6804 Switch provides two VLANs
  - Management VLAN: (rlogin, NFS, etc)
  - Graphics VLAN: (used in the tests reported here)
- Node hardware
  - 1.6 MHz Pentium 4 processor
  - 512 MB RAM
  - 2 NICs operating on separate VLANs
  - 24 Nodes have Intel Pro 1000 NICs on the Graphics VLAN
  - Other nodes have 3COM 3C905C 100 Mbps NICs
  - Maximum common frame size is the 3C905C's 8191

Testbed

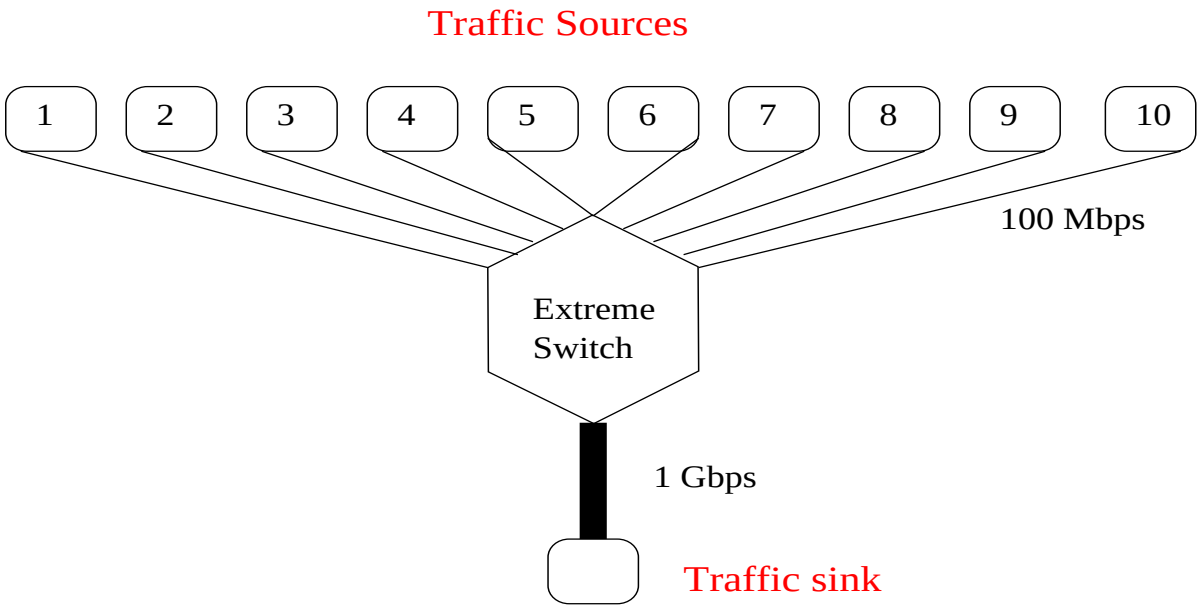
- A 265 node Beowulf type cluster
- Extreme Black Diamond 6804 Switch provides two VLANs
  - Management VLAN: (rlogin, NFS, etc)
  - Graphics VLAN: (used in the tests reported here)
- Node hardware
  - 1.6 MHz Pentium 4 processor
  - 512 MB RAM
  - 2 NICs operating on separate VLANs
  - 24 Nodes have Intel Pro 1000 NICs on the Graphics VLAN
  - Other nodes have 3COM 3C905C 100 Mbps NICs
  - Maximum common frame size is the 3C905C's 8191
- Node software
  - RedHat 7.1 Linux
  - Linux kernel version 2.4.7

UDP Workload



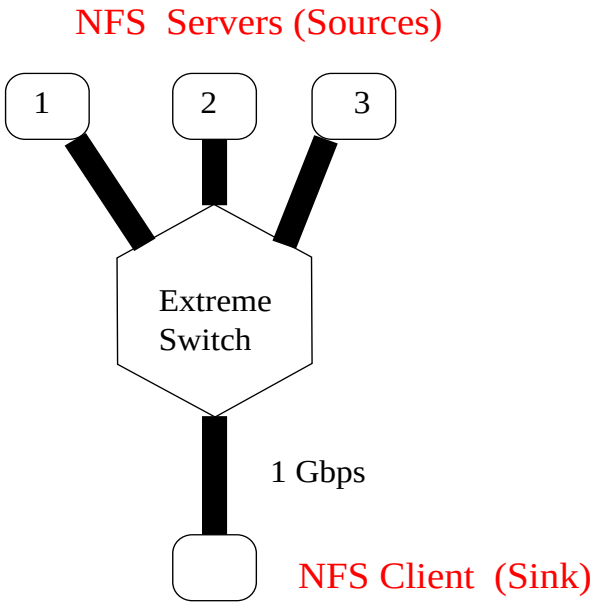
- Simplex flow
- Each sender sends approximately 1.5 GB
- No observed loss
- APDU size is max supported by path MTU

TCP Workload



- Simplex flow
- Each sender sends approximately 1.5 GB
- No observed loss
- APDU size fixed at 16000 bytes

NFS Workload



- Clients and servers all use 1 Gbps NICs
- 3 client processes run on NFS Client
- Each process reads the entire 1.67 GB /usr tree on one server
- Unlike UDP and TCP workloads disk speed *is* a factor

Operational parameters

Two parameters were varied during the testing

- MTU (NPDU size) $\in \{1500, 3000, 4500, 6000\}$
- RxAbsIntDelay $\in \{10, 20, 40, 80, 160, 240, 320, 480, 640\}$
- Interrupt coalescing delays are in units of 1.024μ seconds

Operational parameters

Two parameters were varied during the testing

- MTU (NPDU size) $\in \{1500, 3000, 4500, 6000\}$
- RxAbsIntDelay $\in \{10, 20, 40, 80, 160, 240, 320, 480, 640\}$
- Interrupt coalescing delays are in units of 1.024μ seconds

Other parameters were set to ensure

- Interrupt coalescing would occur
- Tx interrupts would *not* occur
- RxAbsIntDelay would determine packets per interrupt

TxIntDelay	1400
TxAbsIntDelay	65000
RxDescriptors	256
TxDescriptors	256
RxIntDelay	1200

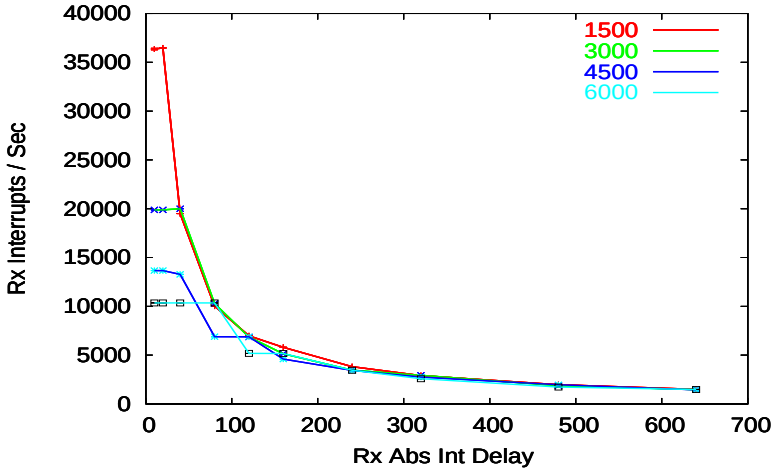
Metrics

Metrics were captured by a customized version 5.2.39 of Intel's e1000 device driver. Metrics reported in the paper include:

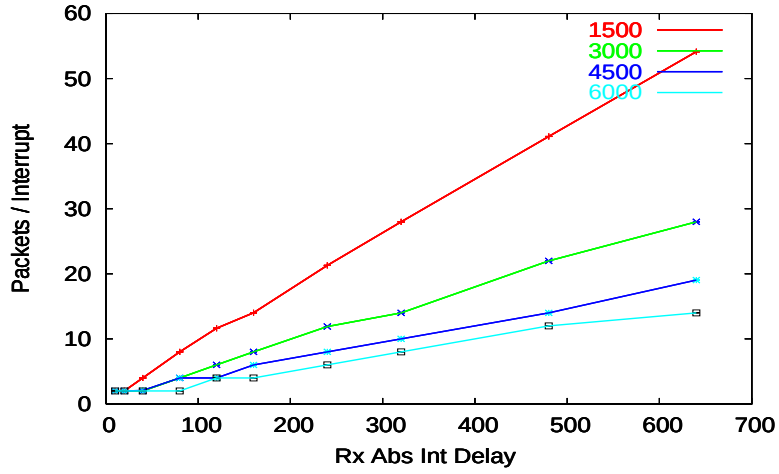
- Interrupts per second
- Packets processed per interrupt
- Throughput in Mbps
- System mode (supervisor state) CPU utilization
- System mode (supervisor state) CPU time

The NFS workload shows that CPU utilization can be a misleading indicator of system performance.

UDP Results

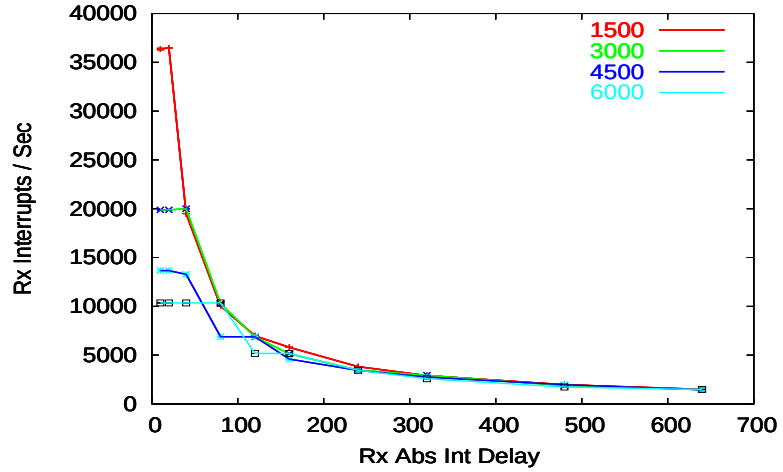


Interrupt Rate

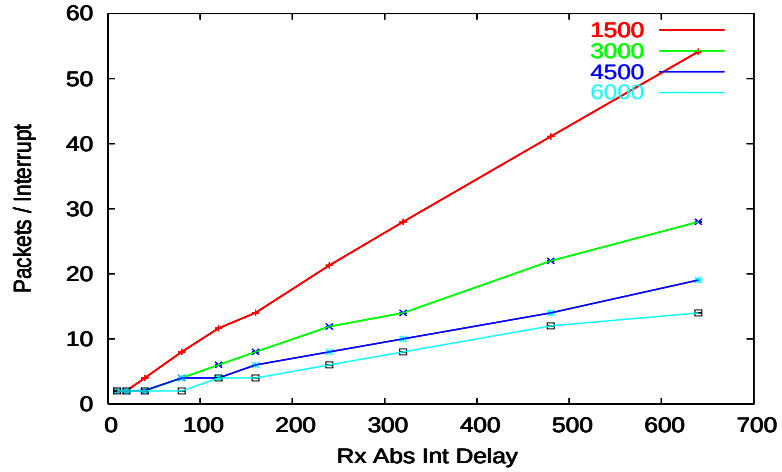


Packets per Interrupt

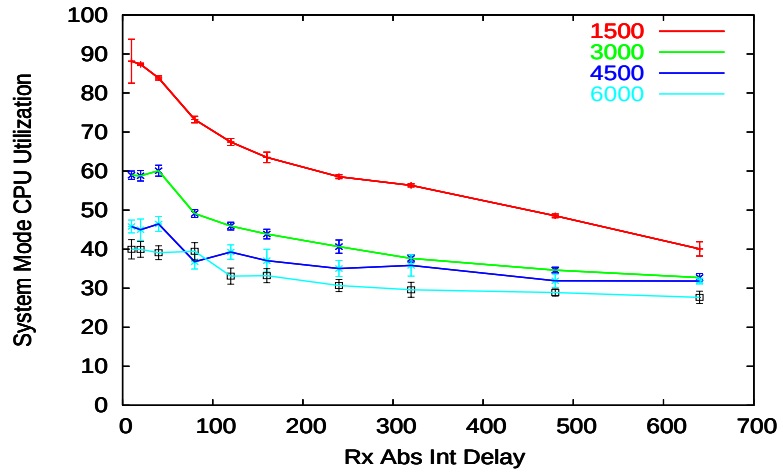
UDP Results



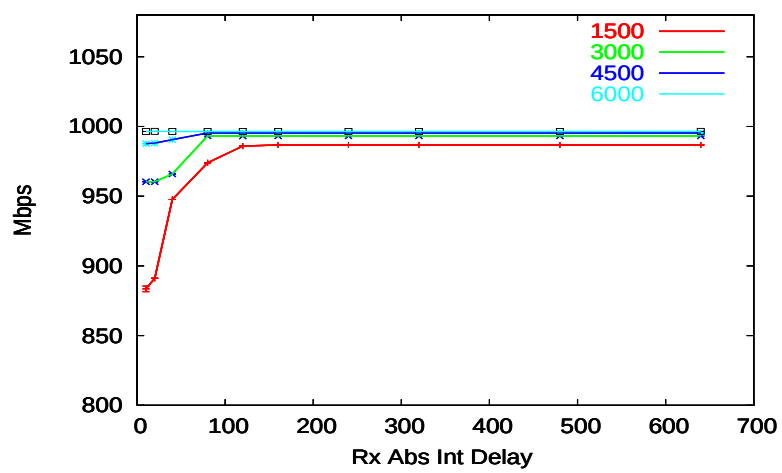
Interrupt Rate



Packets per Interrupt



System Mode CPU utilization



MAC Throughput in Mbps

Observations on the UDP workload

- Increasing frame size is more important than interrupt coalescing in increasing throughput and reducing overhead.

Observations on the UDP workload

- Increasing frame size is more important than interrupt coalescing in increasing throughput and reducing overhead.
- Maximum throughput is reached with RxAbsIntDelay <= 160.

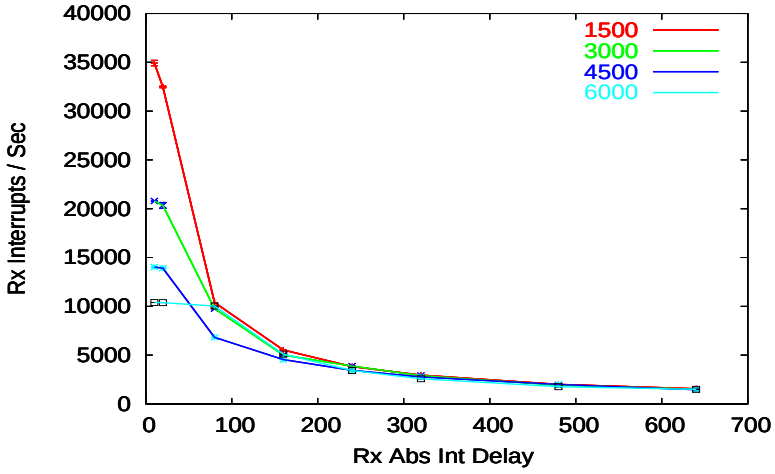
Observations on the UDP workload

- Increasing frame size is more important than interrupt coalescing in increasing throughput and reducing overhead.
- Maximum throughput is reached with RxAbsIntDelay <= 160.
- Reduction in CPU util can be seen for RxAbsIntDelay <= 2000.

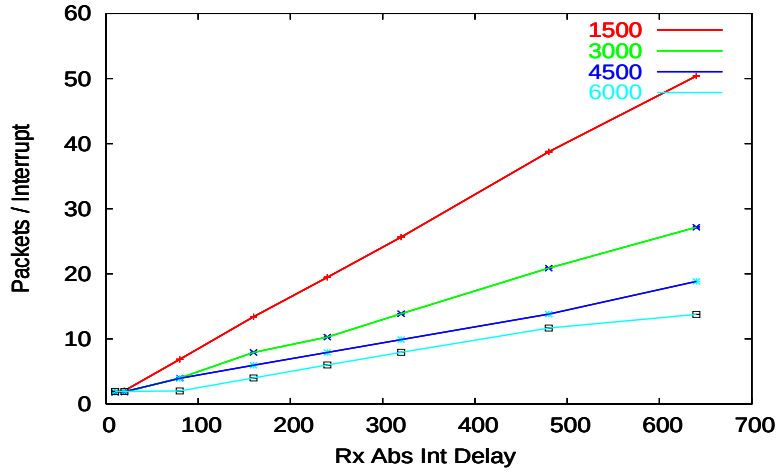
Observations on the UDP workload

- Increasing frame size is more important than interrupt coalescing in increasing throughput and reducing overhead.
- Maximum throughput is reached with RxAbsIntDelay <= 160.
- Reduction in CPU util can be seen for RxAbsIntDelay <= 2000.
- Increasing RxAbsIntDelay can never harm the throughput of an unacknowledged packet flow.

TCP Results

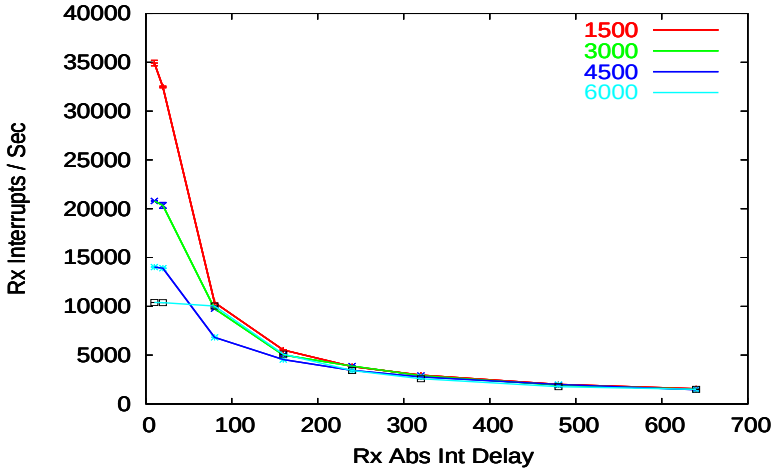


Interrupt Rate

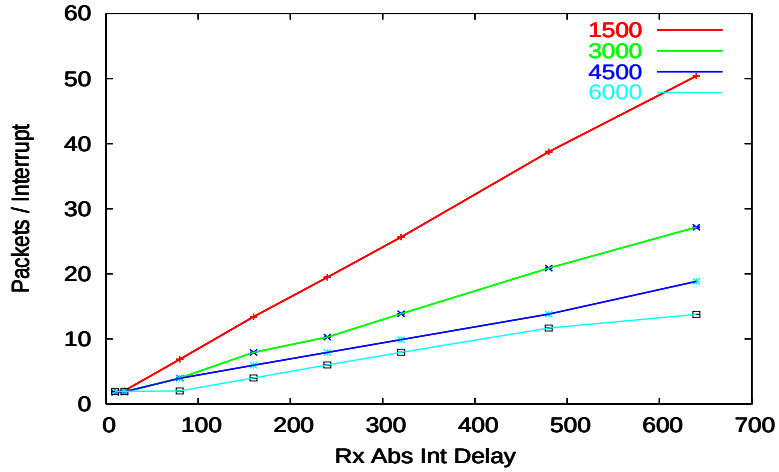


Packets per Interrupt

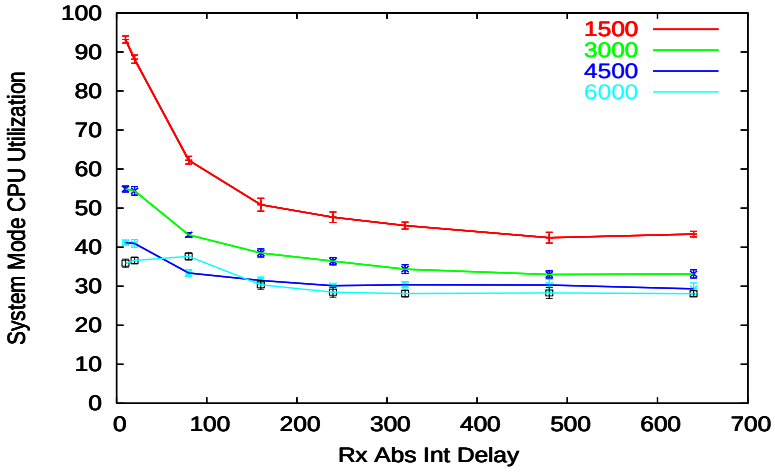
TCP Results



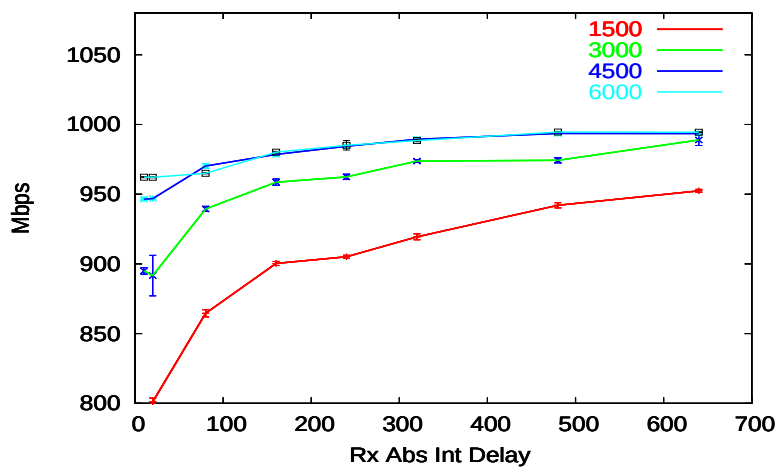
Interrupt Rate



Packets per Interrupt



System Mode CPU utilization



MAC Throughput in Mbps

Observations on the TCP workload

- Throughput grows more slowly with increasing RxAbsIntDelay than was the case with UDP

Observations on the TCP workload

- Throughput grows more slowly with increasing RxAbsIntDelay than was the case with UDP
- Minimum CPU util is achieved with $RxAbsIntDelay \approx 480$.

Observations on the TCP workload

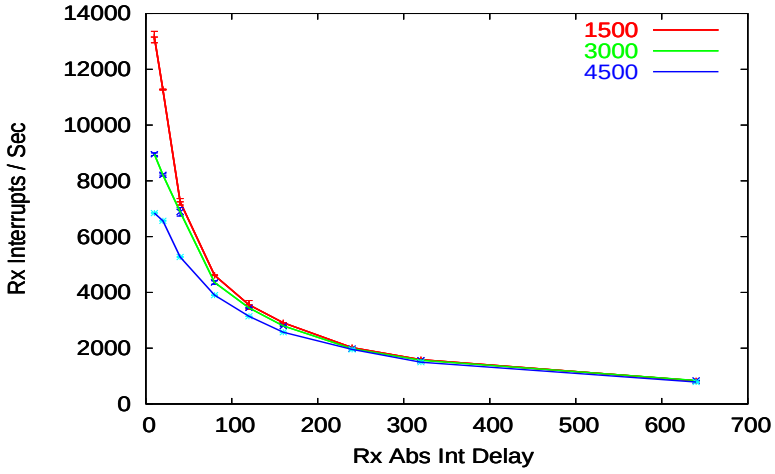
- Throughput grows more slowly with increasing RxAbsIntDelay than was the case with UDP
- Minimum CPU util is achieved with $RxAbsIntDelay \approx 480$.
- Maximum link layer throughput
  - $> 993 Mbps$ for large MTUs
  - $\approx 970 Mbps$ for MTU = 1500
- Maximum throughput is achieved with
  - $RxAbsIntDelay \approx 480$ for large MTUs.
  - $RxAbsIntDelay \approx 960$ for small MTUs.

Observations on the TCP workload

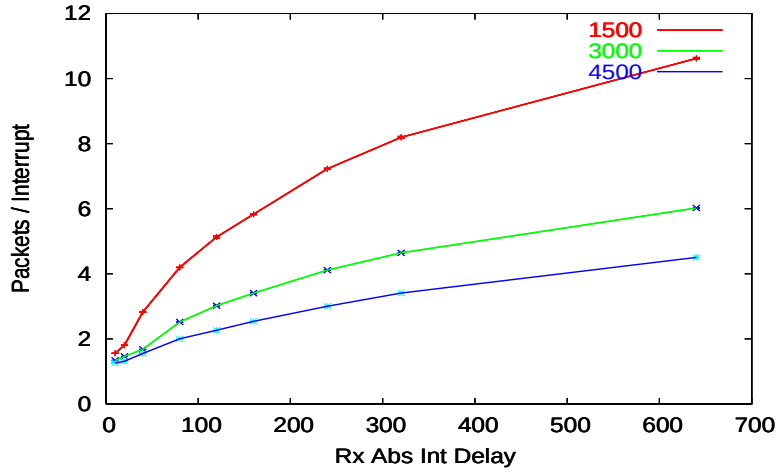
- Throughput grows more slowly with increasing RxAbsIntDelay than was the case with UDP
- Minimum CPU util is achieved with $RxAbsIntDelay \approx 480$.
- Maximum link layer throughput
  - $> 993 Mbps$ for large MTUs
  - $\approx 970 Mbps$ for MTU = 1500
- Maximum throughput is achieved with
  - $RxAbsIntDelay \approx 480$ for large MTUs.
  - $RxAbsIntDelay \approx 960$ for small MTUs.

For the both the 10 sender UDP and TCP workloads near maximal throughput and minimum CPU usage can be obtained with $RxAbsIntDelay \approx 640$

NFS Results

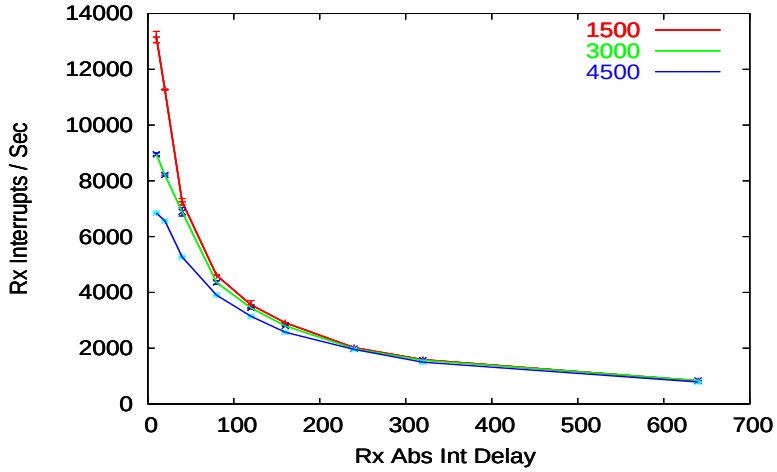


Interrupt Rate

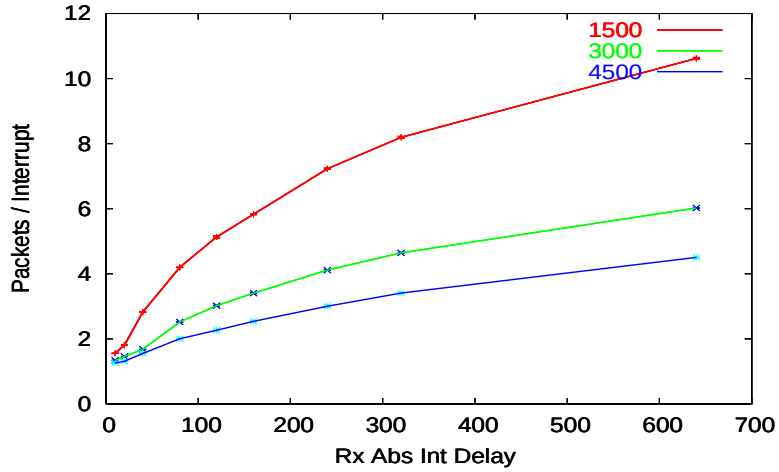


Packets per Interrupt

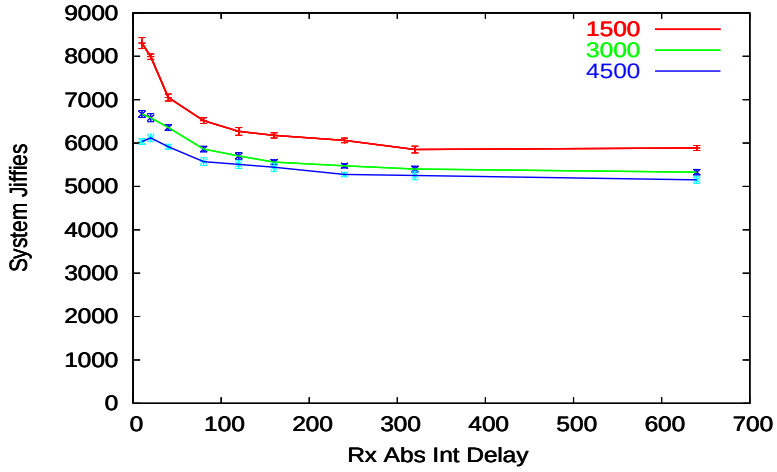
NFS Results



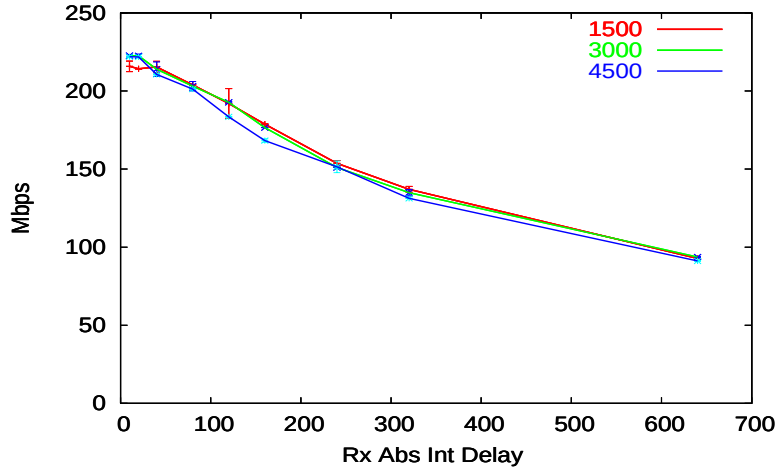
Interrupt Rate



Packets per Interrupt



System Mode CPU consumption



MAC Throughput in Mbps

Observations on the NFS workload

- NFS is a Request - Response protocol
  - Pipelined bursts of arrivals are absent
  - Packets / interrupt no longer grows linearly with RxAbsIntDelay

Observations on the NFS workload

- NFS is a Request - Response protocol
  - Pipelined bursts of arrivals are absent
  - Packets / interrupt no longer grows linearly with RxAbsIntDelay
- Maximum throughput is achieved with
  - $RxAbsIntDelay \approx 40$ for large MTUs.
  - $RxAbsIntDelay \approx 20$ for small MTUs.
  - Rapid throughput decay occurs for $RxAbsIntDelay > 80$

Observations on the NFS workload

- NFS is a Request - Response protocol
  - Pipelined bursts of arrivals are absent
  - Packets / interrupt no longer grows linearly with RxAbsIntDelay
- Maximum throughput is achieved with
  - $RxAbsIntDelay \approx 40$ for large MTUs.
  - $RxAbsIntDelay \approx 20$ for small MTUs.
  - Rapid throughput decay occurs for $RxAbsIntDelay > 80$
- CPU utilization decays with increasing RxAbsIntDelay because the throughput decay is causing less work per unit time to be done.
- Minimal system mode CPU usage is achieved for $RxAbsIntDelay \approx 320$.
- Unlike the UDP and TCP workloads, there is no sweet spot in which both throughput and CPU usage are near optimal.

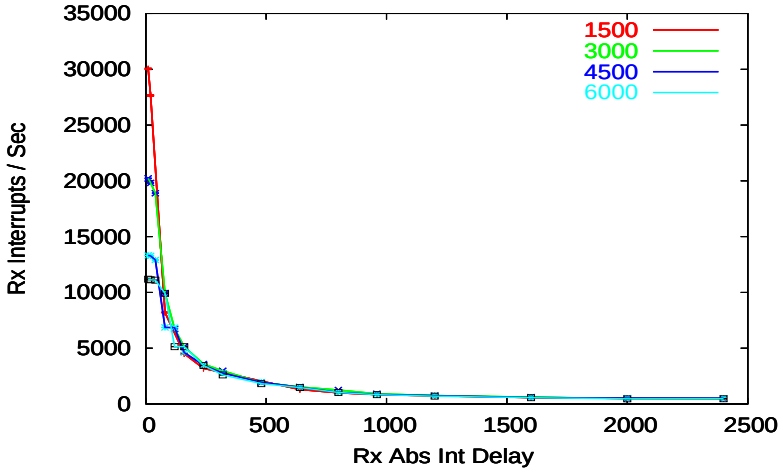
The wrong conclusions to draw

- For UDP and TCP use relatively large RxAbsIntDelay
- For request/response protocols use minimal RxAbsIntDelay

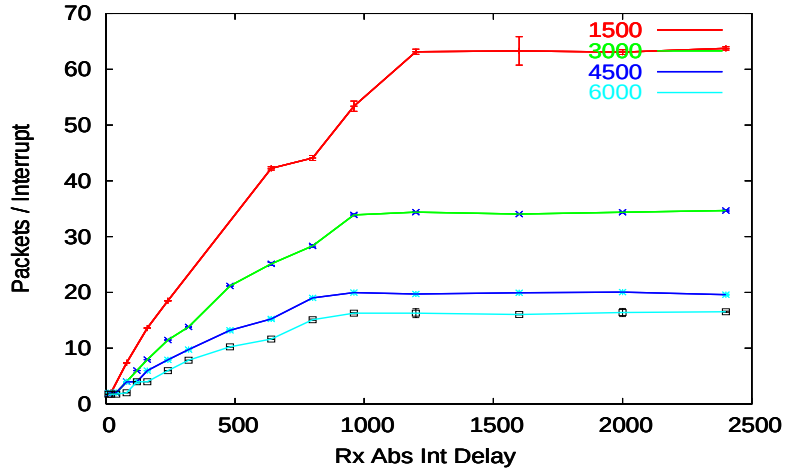
The actual situation is considerably more complex. Even when all nodes reside on a Gigabit LAN the optimal configuration is a function of:

- the number of active senders;
- the rate at which they are sending;
- for TCP senders, min(cwin, offered window).

Three TCP Senders on Gigabit links

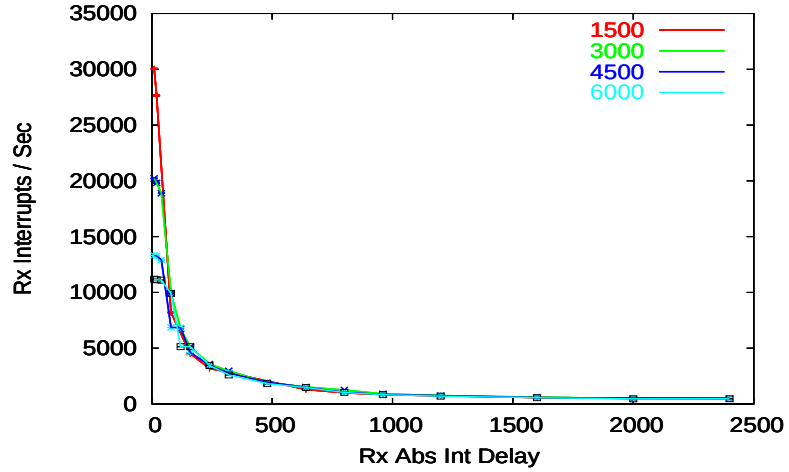


Interrupt Rate

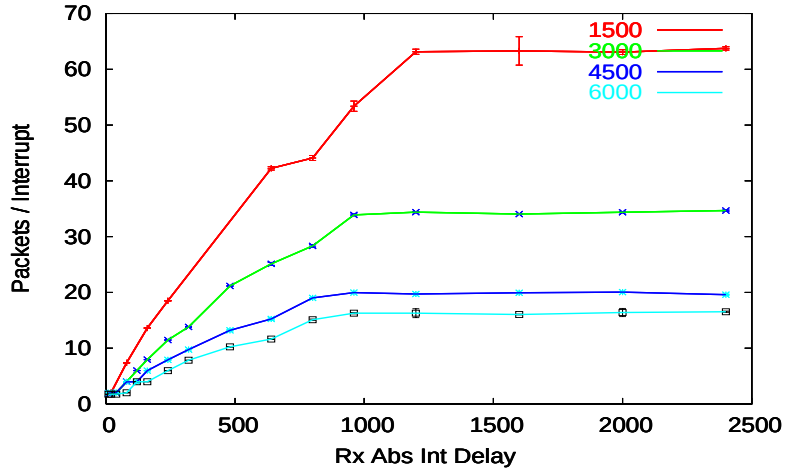


Packets per Interrupt

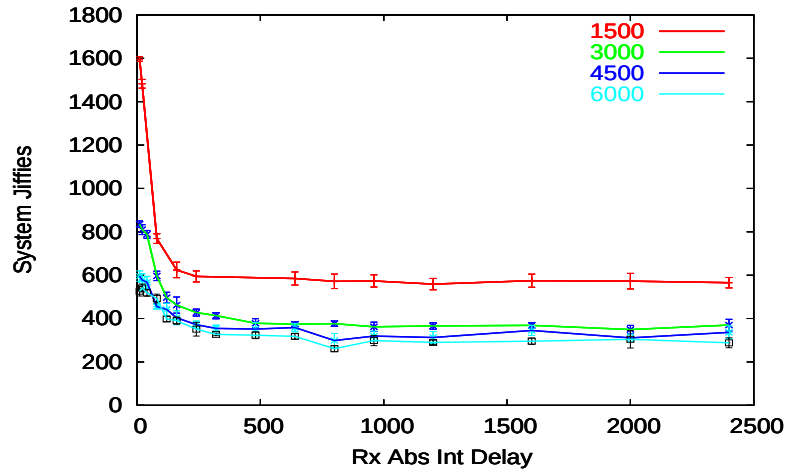
Three TCP Senders on Gigabit links



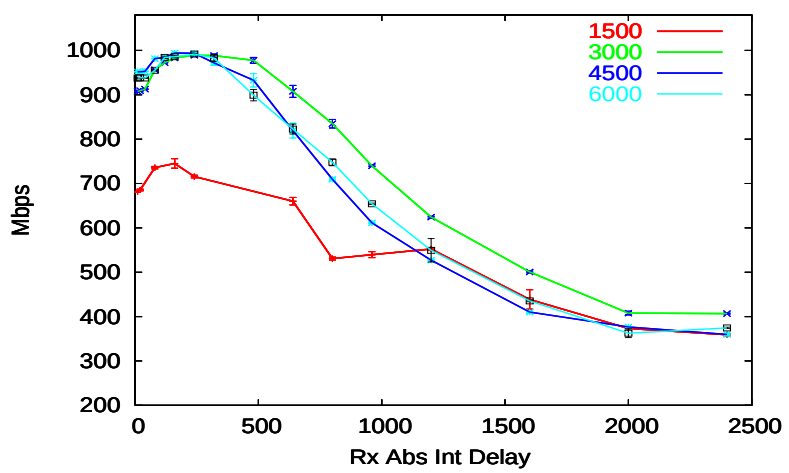
Interrupt Rate



Packets per Interrupt



System Mode CPU consumption



MAC Throughput in Mbps

Observations on the 3 sender TCP workload

- High thoughput sweetspot much narrower than with 10 senders
- Throughput decay occurs:
  - for $RxAbsIntDelay > 320$ in 3 sender workload
  - for $RxAbsIntDelay > 2000$ in 10 sender workload

Observations on the 3 sender TCP workload

- High thoughput sweetspot much narrower than with 10 senders
- Throughput decay occurs:
  - for $RxAbsIntDelay > 320$ in 3 sender workload
  - for $RxAbsIntDelay > 2000$ in 10 sender workload
- Good CPU util is achieved with $RxAbsIntDelay \approx 480$.

Observations on the 3 sender TCP workload

- High thoughput sweetspot much narrower than with 10 senders
- Throughput decay occurs:
  - for $RxAbsIntDelay > 320$ in 3 sender workload
  - for $RxAbsIntDelay > 2000$ in 10 sender workload
- Good CPU util is achieved with $RxAbsIntDelay \approx 480$.
- But unlike the NFS workload 480 provides an operating point at which good throughput and CPU usage can be obtained.

TCP in a Gigabit Ethernet Environment

- For good TCP performance TCP senders must not block on full *ownd* or *cwnd*.
- If *acks* are excessively delayed senders will block.
- Interrupt coalescing does delay packet delivery and thus also *acks*.

TCP in a Gigabit Ethernet Environment

Therefore,

- Having a large number of senders is good.
- Having large sender buffer space (large *ownd*) is good but,
- an offered load that exceeds receiver link capacity results in a small *cwnd* which is bad.

TCP in a Gigabit Ethernet Environment

Therefore,

- Having a large number of senders is good.
- Having large sender buffer space (large *ownd*) is good but,
- an offered load that exceeds receiver link capacity results in a small *cwnd* which is bad.

Furthermore,

- Interrupt coalescing can lead to both
  - ack compression
  - ack rate reductions (e.g. 1 ack per 5 or 6 packets)
- These are known to be bad in a WAN but
- produce no apparent ill effects in a LAN.

A model for CPU usage

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

- α = Per megabyte cost of checksumming and copying
- β = Per thousand packet cost of protocol operations
- γ = Per thousand interrupt cost of context switching
- For TCP $\alpha \approx 0.1, \beta \approx 0.4, \gamma \approx 1.4$ jiffies

A model for CPU usage

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

- α = Per megabyte cost of checksumming and copying
- β = Per thousand packet cost of protocol operations
- γ = Per thousand interrupt cost of context switching
- For TCP $\alpha \approx 0.1, \beta \approx 0.4, \gamma \approx 1.4$ jiffies

Suppose one MB of data is to be transfered using application payload per packet = MTU and packets per interrupt = PPI.

Then $T = \alpha + \frac{1000}{MTU}\beta + \frac{1000}{MTU \times PPI}\gamma$

- $PPI << MTU$
- PPI impacts only the last term
- MTU plays a much more significant role in reducing CPU usage

In summary

- Increasing MTU does not hurt throughput or increase CPU usage

In summary

- Increasing MTU does not hurt throughput or increase CPU usage
- Increasing RxAbsIntDelay does not increase CPU usage

In summary

- Increasing MTU does not hurt throughput or increase CPU usage
- Increasing RxAbsIntDelay does not increase CPU usage
- Increasing RxAbsIntDelay can severely degrade throughput but
- The onset and magnitude of the degradation is strongly workload dependent.

In summary

- Increasing MTU does not hurt throughput or increase CPU usage
- Increasing RxAbsIntDelay does not increase CPU usage
- Increasing RxAbsIntDelay can severely degrade throughput but
- The onset and magnitude of the degradation is strongly workload dependent.

These results suggest RxAbsIntDelay be adjusted dynamically

- Intel provides a dynamic mode

In summary

- Increasing MTU does not hurt throughput or increase CPU usage
- Increasing RxAbsIntDelay does not increase CPU usage
- Increasing RxAbsIntDelay can severely degrade throughput but
- The onset and magnitude of the degradation is strongly workload dependent.

These results suggest RxAbsIntDelay be adjusted dynamically

- Intel provides a dynamic mode
- but it provided lower throughput and higher CPU usage for all MTUs on the 10 TCP sender workload.

In summary

- Increasing MTU does not hurt throughput or increase CPU usage
- Increasing RxAbsIntDelay does not increase CPU usage
- Increasing RxAbsIntDelay can severely degrade throughput but
- The onset and magnitude of the degradation is strongly workload dependent.

These results suggest RxAbsIntDelay be adjusted dynamically

- Intel provides a dynamic mode
- but it provided lower throughput and higher CPU usage for all MTUs on the 10 TCP sender workload.
- More research is needed ... especially as 10 Gbit NICs become common.