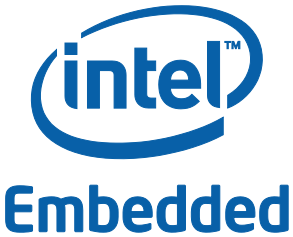


White Paper

**Joseph Gasparakis**
Performance Products Division,
Embedded & Communications Group,
Intel Corporation

**Peter P Waskiewicz, Jr.**
LAN Access Division,
Data Center Group,
Intel Corporation

# Design considerations for efficient network applications with Intel® multi-core processor-based systems on Linux*

July 2010

Order: 324176-001US

# Executive Summary

In this paper we present the outcome of research conducted to determine the most efficient ways to develop networking applications using an Intel® multi-core processor-based system and multi-queue capable network interfaces running Linux*. We analyze the hardware platform attributes, the software aspects, and provide clear guidelines for the reader about performance pitfalls. We also include pointers for improving performance when designing network applications to run at speeds of 10 Gb/s using Linux. Finally, we present the results for IP forwarding (routing) with and without the optimizations mentioned in this paper to illustrate the benefits of applying these findings.

# Contents

# Introduction

This paper presents the outcome of research and analysis for developing performance networking applications in an Intel® architecture-based multi-core system with multi-queue network controllers using the Linux* operating system. This white paper consists of the following sections:

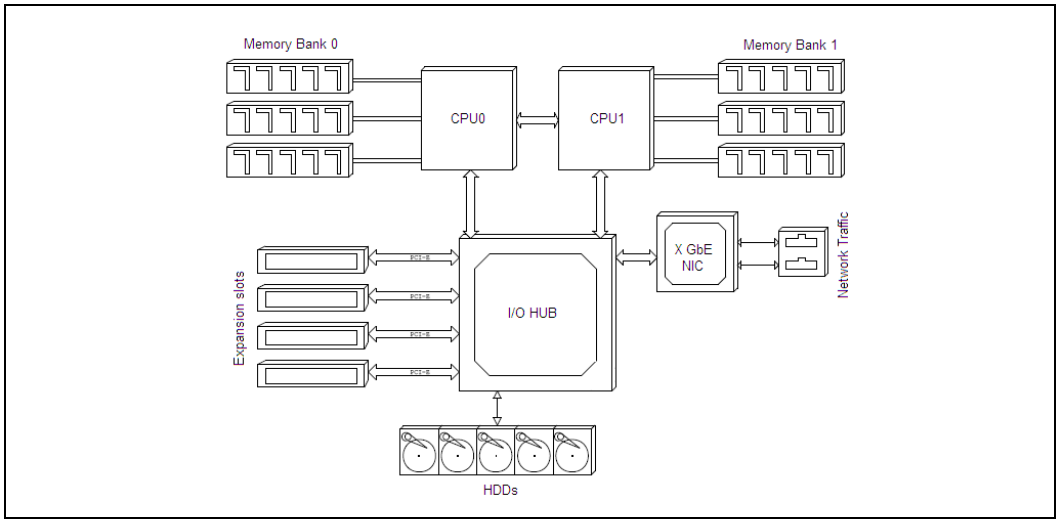
- Background discusses hardware and software concepts that the reader should be familiar in the scope of this work.
- Optimization Analysis presents the hardware technologies and software techniques that play a significant role in the performance of a networking application.
- Administrative Linux Networking Considerations explains how the technologies and techniques mentioned above can be combined from a system administration view. This section describes the hands-on part of the implementation.
- Identifying the Bottlenecks describes a set of tools that we found useful in our analysis and gives a brief description of how the tools can be used.
- Experimental Numerical Data Based on IP Forwarding presents the results for an optimized and a non optimized system doing routing (IP forwarding) in the Linux kernel in order to showcase and stress the importance of all the performance attributes. At the very end of the section, we discuss the results.
- Conclusion and References are also provided.

# Background

## Hardware: Modern Non-Uniform Memory Access computer architectures

In modern Non-Uniform Memory Access (NUMA) computer architectures, two or more CPU dies are placed on the board as shown in Figure 1. In Intel Architecture, these dies communicate with each other using the Intel® QuickPath Interconnect (Intel® QPI) bus [1].

**Figure 1. Modern NUMA Computer Architecture**




Another feature of this architecture is that the memory controller is embedded in the same package as the CPU. For a multi-die system, that would mean that there is typically more than one memory controller. We say typically as there is always the choice of not populating the memory bank of a controller. In any case, in a multi-die system, the memory allocations need to be carefully planned, as there are significant performance penalties when accessing memory allocated on the remote node. These penalties can include: saturating the Intel® QPI bus, thrashing the caches between CPUs, and adding extra latency to the memory fetch.

The Linux kernel provides APIs for the developer to choose their node and to have better control on how and where the memory is allocated. [2]

## Software: Multi-core and multi-queue in the Linux kernel network stack

Versions of the Linux kernel prior to 2.6.21 were not multi-queue aware. Network frames in these versions are received and processed in a serial manner. As an example, the IP Forwarding data flow diagram for these kernels is shown in Figure 2.

In kernel version 2.6.21, receive side multi-queue support was added. Later on in version 2.6.23, the kernel became multi-queue aware on the transmit side also, parallelizing a great deal of the stack and improving the data path to and from the Linux kernel as shown in Figure 3.

However as of the current version 2.6.31, the kernel still does not fully utilize parallel processing for the whole data path as noted by Luca Deri in [3]. When attempting to introduce sufficient parallelism into the kernel, careful design is needed to maintain consistency in support of all protocols, and to avoid introducing packet reordering, which can be catastrophic.

**Figure 2. Legacy IP Forwarding in the Linux Kernel**

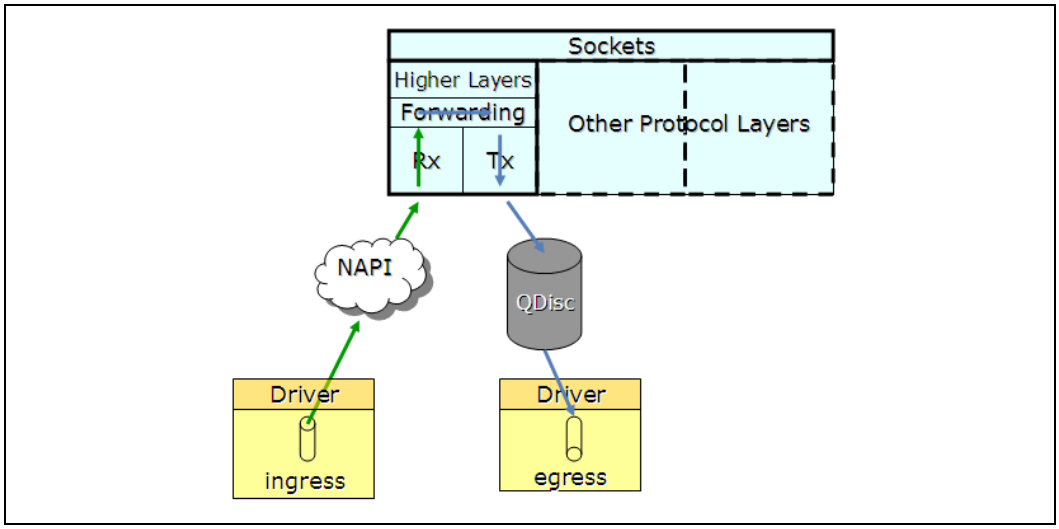
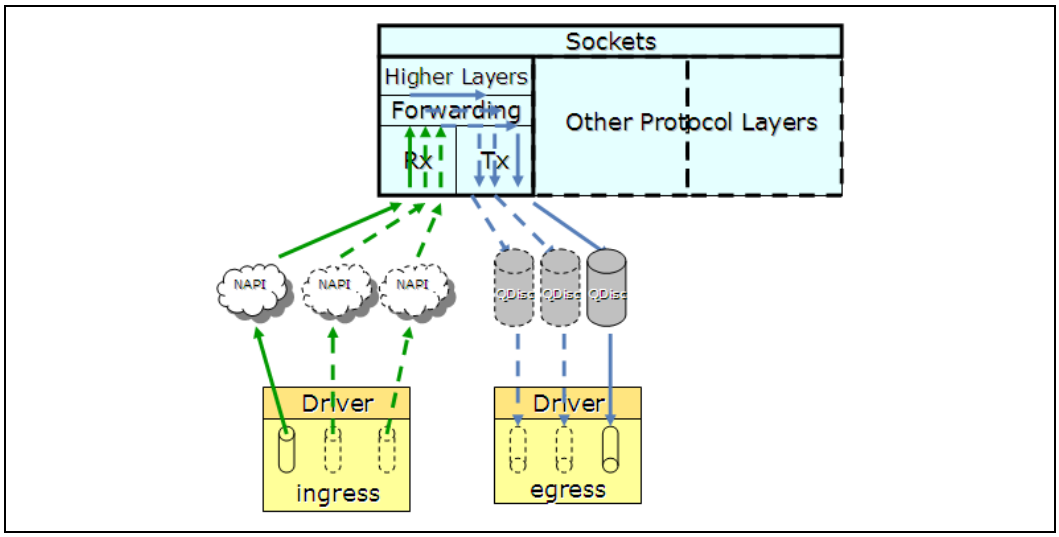



**Figure 3. Multi-queue aware IP forwarding in the Linux Kernel Version 2.6.21 and later**




# Optimization Analysis

In this section we expand on the features that Intel® hardware can provide and how they can be exploited in a Linux context. It is important for the reader to keep in mind that not all of the features in this section will yield a performance boost on all types of applications. In fact, some types of applications can be negatively impacted by enabling some of these features,

whereas other applications can benefit. This is why it is important to understand what the features are before the design phase. It is also a good idea to prototype and go through some performance analysis stages before committing to a particular design.

# Hardware Platform Architecture Considerations

It is very important to analyze the hardware architecture before anything else. The hardware should not be the source of degraded performance; therefore data paths and their bus throughputs should be considered. It is worth mentioning that due to bus protocols (such as in the PCIe protocol), the throughput of certain channels can, in fact, be lower than in a specification in terms of actual data throughput. As an example, the protocol of the PCIe requires encapsulation of the messages and preamble which takes part of the bus throughput.

## CPU Architecture Considerations

Several features of the CPU can impact the performance of a networking application. Power management features in general can reduce the performance under certain circumstances. If the CPU is in C1 state, for example, although transitioning into C0 state is almost instantaneous, in reality that can negatively affect the processing of the first packets due to cache latencies.

Similarly, Intel® Turbo Boost Technology may improve performance or may degrade it, depending on the application. However, in full multi-queue processing loads, more than one CPU core will be online, thus negating the effect of Intel® Turbo Boost Technology mode.

Intel® Hyper-Threading Technology (Intel® HT Technology) also needs to be considered carefully as in some cases it can cause cache thrashing in networking applications. Furthermore, Intel® HT Technology can also impact the efficiency of interrupt vectors assigned to the logical CPU core. Care must be taken to assign interrupts correctly to both the physical core and its child hyper-thread. Later in this paper we discuss some pointers on Intel® HT Technology.

# Network Controller Features

## Input/Output Acceleration Technology

Intel® I/O Acceleration Technology (Intel® I/OAT) [4] [5] is a set of technologies that contributes to increased performance by reducing the CPU cycles spent on input and output related procedures. The technology set is described below.



## Intel® QuickData Technology

Intel® QuickData Technology offloads input and output data copies to the chipset, allowing the CPU to spend its cycles on other functions. For more details, the reader is encouraged to read reference [6].

## Direct Cache Access (DCA)

Direct Cache Access is a platform-wide method to deliver inbound I/O data directly into the processor's caches, providing a significant performance increase [7]. The design goals of DCA are 1) timely availability of the data in cache leading directly to a lower average memory latency and 2) reduction in required memory bandwidth.

Using DCA can produce varying results based on the type of workload being processed, plus what other workloads are on the system. If the CPU dcaches are full, then DCA will cause the CPU to evict something from the dcache, which could produce unintended cache-thrash. However, for high-priority network traffic flows that require lower latency, this mechanism can be very useful to getting CPU time immediately.

DCA is currently an Intel® technology, but has generated enough interest that the PCI-SIG incorporated a native DCA-like functionality for PCI Express 3.0, called TPH. It will allow tags, similar to DCA, to be generated to create cache hints for the CPUs.

## Header split and replication

Depending on the type of application, a performance boost can be achieved with header replication or header split techniques. These techniques are described in detail in the Intel® 82576 Gigabit Ethernet Controller Datasheet [8], specifically in section 7.1.9. In brief, this feature consists of splitting or replicating packet's header to a different memory space. This is accomplished by the Ethernet controller issuing DMA requests for both the header buffer and the payload into separate locations. This helps the host to fetch headers only for processing: headers are replicated through a regular snoop transaction in order to be processed by the host CPU. This also allows payload buffers to be placed into pages, which is much more efficient to deal with than non-page aligned buffers.

When dealing with larger frames, such as Jumbo Frames, header split techniques are very useful to increase the efficiency of memory allocation and DMA. This way, 9 KB payloads, as an example, can be split into 3 pages, making the DMA across PCI Express much more efficient than trying to push 9 KB in a single request. In addition, it allows the Linux memory manager in the kernel to allocate pages versus large non-aligned buffers, which is also much more efficient.

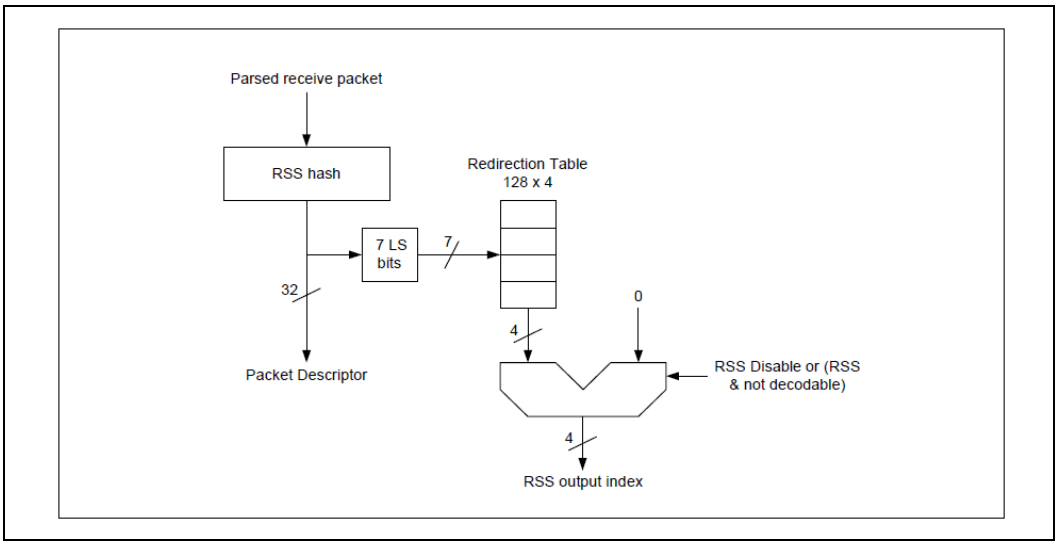
## Extended Message Signaled Interrupts

As part of PCI 2.2 that was later inherited by PCI Express (PCIe), interrupts are distributed as messages in the bus, rather than being dedicated signals to the CPU. Extended Message Signaled Interrupts (MSI-X) are I/O interrupts that can be distributed to multiple CPU cores allowing the data to be shared and processed amongst the desired cores. In Linux, interrupt distribution can be done through SMP affinity as described in Interrupt Affinities and Queues on page 11.

## Receive Side Scaling

Receive Side Scaling (RSS) is a mechanism to distribute received packets into several descriptor queues. The kernel can then assign each queue to a different CPU core based on the SMP affinity defined by the user. Using the Intel® 82599 10 GbE Controller as an example, when a packet is received, the header fields such as IP addresses, port, and protocol number are fed to a hash operation. Then the 32 bit result is fed into the packet receive descriptor and the seven least significant bits are used as an index into a 128 entry redirection table. Each entry provides a 4-bit RSS output index. Figure 4 describes RSS in terms of a block diagram.

**Figure 4. RSS Block Diagram**




## Rx Queue Assignment

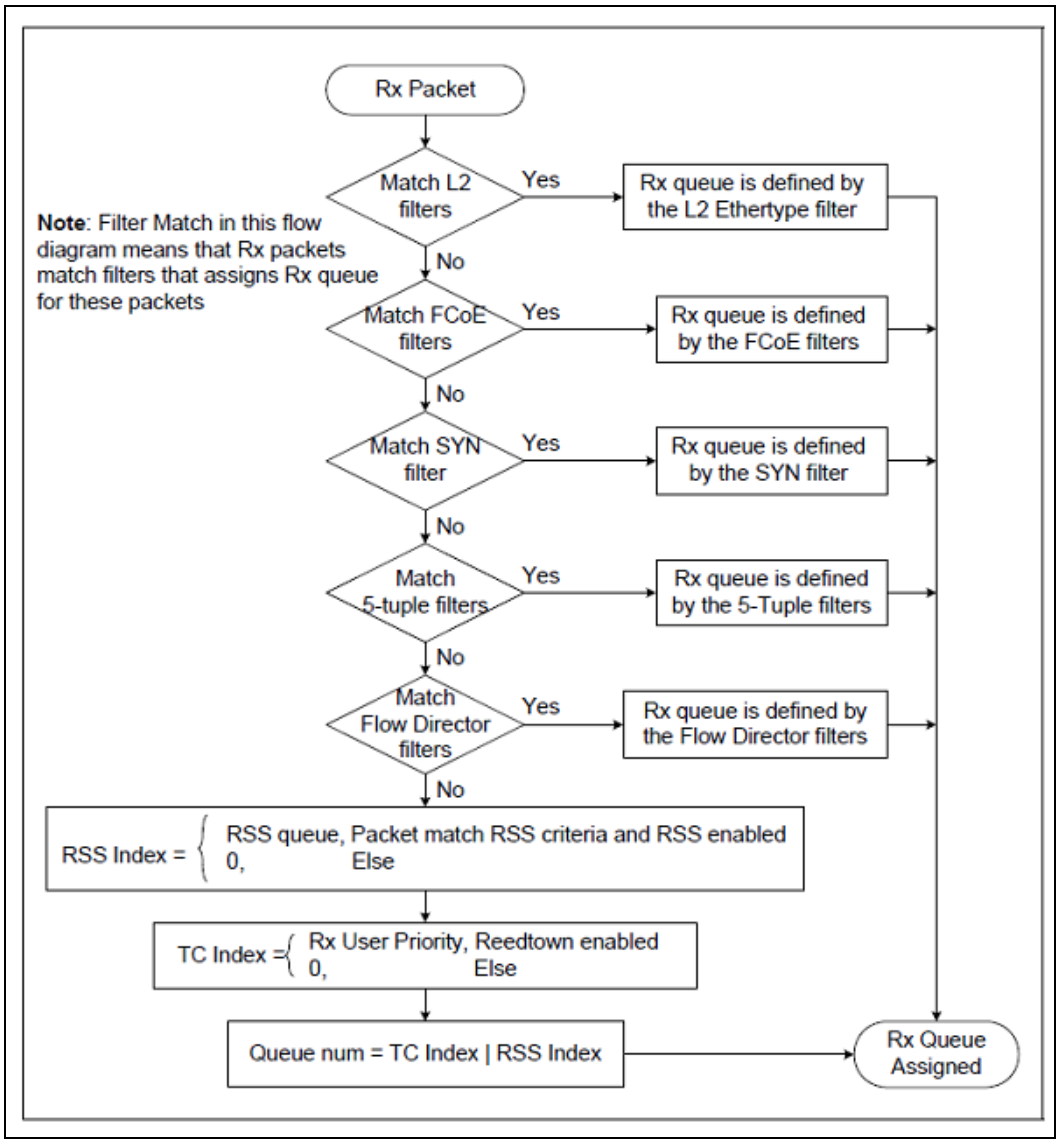
Another feature that a network controller might have is the ability to do Rx queue assignment. An example of a controller with this capability is the Intel® 82599 10 GbE Controller.

The Rx Queue assignment feature allows the software to determine which queue to place the packet in after processing, using hardware filters (similar

to the RSS feature). With the help of IRQ affinity, again, the packet is directed to the desired core or cores. Figure 5 describes the different stages of filtering for queue assignment. Using this feature, the software can control what packet flows go to which core. Then a thread can be run on that core that is specific to that type of network flow.

More details on this feature can be found in the Intel® 82599 10 GbE Controller datasheet. [9]

**Figure 5. Rx Queuing Flow in Intel® 82599 10 GbE Controller**



### Receive Side Coalescing

Receive Side Coalescing (RSC) aggregates packets from the same TCP/IP flow into a larger packet reducing the per-packet processing overhead in the CPU. The limitation of this technology is that it can be used only for TCP/IP network data stream at this time. This is also a technology that isn't compatible with bridging or routing implementations.

### Low Latency Interrupts

Low Latency Interrupts allow the interrupt interval times to be set in such a way that it will allow the right latency sensitivity for the data. This minimizes context switches due to interrupts. A real-time related networking application can benefit from this feature.

## Linux Kernel Considerations

One of the advantages of the Linux kernel is that it offers the option to add support for specific hardware to the system, allowing for a more flexible kernel that will not spend cycles trying to support devices that are not present. Absolutely necessary functions include support for the networking stack, the memory controller, and memory management related modules. For instance, in a purely networking design the sound support or the USB support might be completely removed. Unless it is required, removing IPv6 support from the kernel can also positively impact the performance of the system.

### Interrupt Affinities and Queues

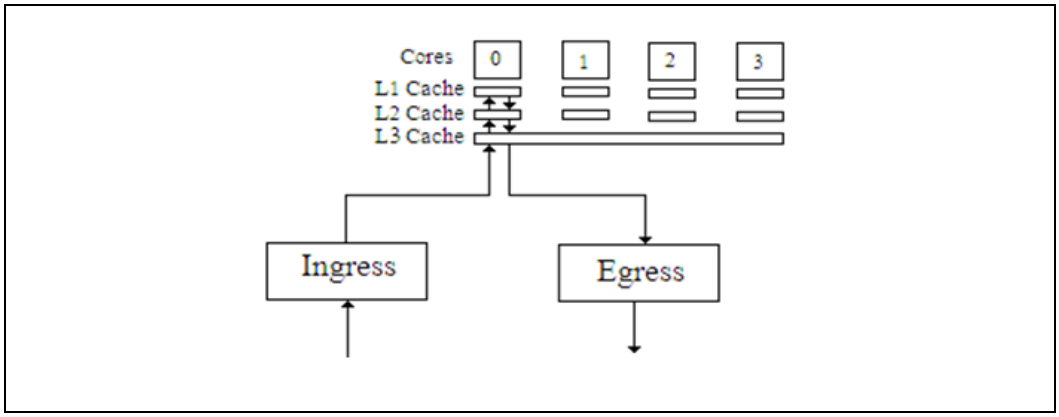
One important factor in high performance IP forwarding that needs to be taken into account is the interrupt affinity or SMP affinity. By pairing the right interrupts to the right CPU cores, the design can achieve maximum cache coherency or better interrupt spreading. Code that will be released in kernel version 2.6.35 allows drivers to specify their own affinity for each interrupt to achieve this mapping out of the box.

In order to use NAPI (the new API) in the kernel, a driver simply has to register a callback function with the interrupt subsystem. This callback is assigned to each interrupt vector, and should return a CPU mask that is the desired affinity for that interrupt. Using the userspace program *irqbalance*, the affinity will be automatically enforced without user intervention. Support for this functionality exists in the latest *irqbalance* release, version 0.56. Note that running a prior version of *irqbalance* may interfere with the affinities, which may cause undesired results.

### Designing for cache coherency

Designing for cache coherency has the goal of minimizing the cache misses and cache thrashing. Figure 6 shows the diagram for a design with this goal.

**Figure 6. IRQ affinity targeting cache utilization**

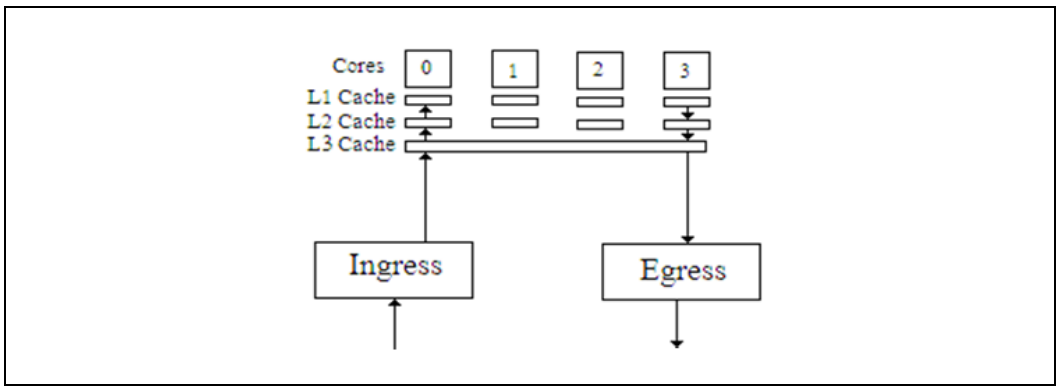



Avoiding cache misses is an important performance factor, and its importance is magnified in a NUMA setup such as the one shown in Figure 1.

## Designing for maximum interrupt spread

Another factor that has impact in the performance of the system is the number of interrupts per CPU core. An interrupt is a very intrusive function that can significantly decrease performance due to the context switches it creates. NAPI is an interrupt mitigation technique that is implemented in the Linux kernel as illustrated in Figure 3. It greatly decreases the number of interrupts that occur in CPU cores. It should be noted that in high throughput scenarios, such as in small packets in 10 GbE, NAPI might not be sufficient. In that case, the design shown in Figure 7 can prove helpful.

**Figure 7. IRQ affinity targeting interrupt spread**




## Discussion on the two designs

The two designs are fundamentally different. The design for cache coherency is focused on keeping the caches per core clean, but takes the impact of having the interrupts handled by the same core. Usually this is the design that proves more efficient; however, there are cases where a design should

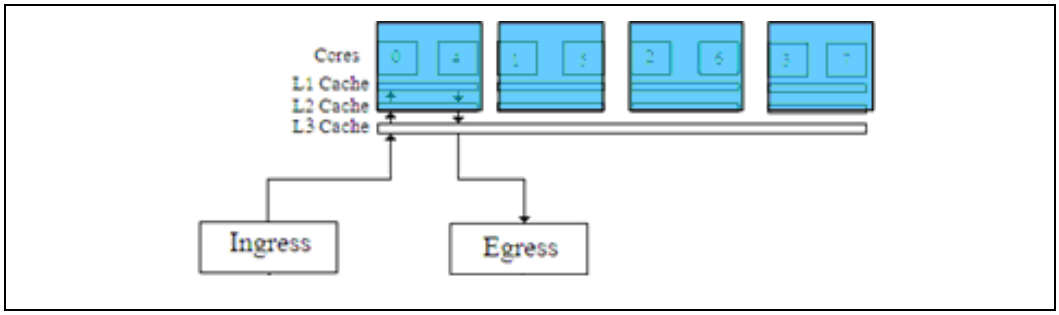
target the interrupt spreading pattern shown in Figure 7. Sometimes it is worth spreading the impact of the context switches, especially if the network traffic is such that the cache misses are expected anyway. The developer needs to make a careful analysis of the design approach, and use prototyping to verify the designs. A combination of good choices between these two designs based on the predictability of the expected traffic and the type of networking application can greatly increase the efficiency of the system.

## Designing for SMT systems

For Simultaneous Multi-Threading (SMT) enabled kernels, the number of the CPU cores reported is twice the number of physical cores. This is because for every physical core present, there is a logical core representing the second hardware thread. The physical cores are enumerated first and then the logical ones. For example, in a quad-core system as shown in Figure 8, the physical cores are represented as core 0, core 1, core 2, and core 3, where the logical ones follow as core 4, core 5, core 6, and core 7.

As each logical core represents the “spare cycles” of a physical core [10], it is wise to also affinitize the interrupts of the physical core to the corresponding logical core. This helps avoiding cache thrashing and the other negative side effects described in previous sections of this document.

**Figure 8. IRQ affinity for SMT systems**



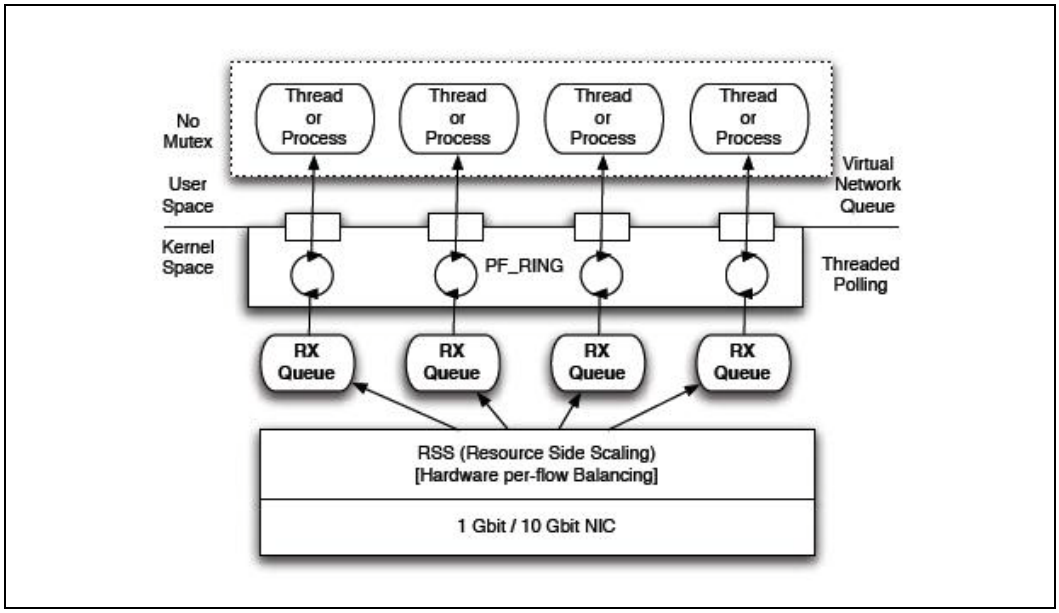

## Kernel vs User space networking applications

Another key decision that impacts performance is which part of the software stack runs inside the kernel versus that which runs in user space. The general rule of thumb is that modules in kernel context should stay as lightweight as possible. The reasoning is that the kernel’s purpose is to efficiently handle the resources of the system, therefore adding a software module that does a lot of processing would impact kernel efficiency in handling and providing for these resources.

For networking applications in user space, the design needs to take into account the performance penalty of copying buffers to/from user/kernel space. Memory mapped I/O is always a better solution. In applications such as network monitoring, there are other options such as PF_RING [11] which uses memory mapped I/O and pushes packets up to user space through a

ring buffer bypassing the whole network stack. PF_RING could provide a good performance boost for an application that needs only raw packets up in user space, but could be a problem if these packets need to trigger mechanisms in the network stack of the kernel. Lately the PF_RING has been multi-threaded and with the help of TNAPI, as shown in Figure 9, it can be used for very efficient packet transitioning from the wire into the user space [3].

**Figure 9. TNAPI Architecture**




# Administrative Linux Networking Considerations

## General Networking Resources

As described in [12], one of the first things that needs verification is speed and duplexing. Ethtool [13] can be used to provide information about a network interface in a Linux based system, including such information as to whether the queried interface has linked in the desired speed and if it did in full or half duplex.

Maximum Transmission Units (MTU) is another consideration that should be taken into account. If the designed system is to be connected to a network with large MTU, it should have a large MTU as well. This allows the designed system to exchange large frames (Jumbo Frames) significantly reducing the overhead per frame.

Also, increasing network buffers and the amount of unprocessed packets (/proc/sys/net/core/netdev_max_backlog) and/or tuning the window sizes in /proc/sys/net pseudo file system can offer a performance increase, depending on the application. Note that writing values in the /proc file system is only a temporary solution as the values will not be retained after a system reboot. For the changes to become permanent, the sysctl command should be used. Again the reader is encouraged to refer to Chapter 4 of [12] for more details.

### SMP Affinity in Linux

Finally, the SMP affinity of the MSI-X can be determined by echoing a mask in the /proc pseudo file system, specifically in /proc/irq/X/smp_affinity where X is the IRQ number for the specific queue. The IRQ number can be obtained by examining the first column of the /proc/interrupts pseudo file. The mask that is echoed into the appropriate smp_affinity pseudo file describes what core the interrupt can be sent to. Bit 1 of the mask enables or disables core 0, bit 2 enables or disables core 1, bit 3 enables or disables core 2, etc. For example, to bind core 0 to interrupt number 136, use:

```
#echo 0x1 > /proc/irq/136/smp_affinity
```

Similarly, to bind interrupt number 262 to core 6, use:

```
#echo 0x20 > /proc/irq/262/smp_affinity
```

This operation requires root privileges.

***Note:*** The SMP affinities will remain set until the system powers down or resets. After a power cycle or after a reset, the SMP affinity needs to be set again otherwise the interrupt-core binding will be random.

### Driver features and parameters

The drivers in Linux can be insmoded with certain parameters set. The Linux driver for the Intel© 10 Gb/s network controllers (ixgbe) uses several parameters that enable/disable or otherwise affect the features described in the Network Controller Features section in this document.

# Identifying the Bottlenecks

There are many tools that one can use in order to identify bottlenecks in a Linux system, but common logic should be used first: Running unnecessary daemons and services can have a very negative performance impact. Especially running IRQ balancer daemon (*irqbalance*) will cause any SMP affinities to have no impact, thus off-balancing the interrupts and thrashing the caches. Again, in the 2.6.35 version of the kernel with version 0.56 of *irqbalance*, the problem of cache thrashing is removed with automatic (and correct) enforcement IRQ affinities. Also in heavy networking application devices such as routers, running the X server or having a system with



SELinux can have a large negative impact as they use up many resources. Using Netfilter can also have a negative impact. The reader is encouraged to refer to [12] for details. SELinux also might impact the flow of packets depending on its security policies. However, SELinux can greatly help the security of the system and prevent successful attempts of compromising the system. The designer needs to make a decision on the performance versus security tradeoff.

In the /proc pseudo-file system, several performance counters can be read directly. Alternatively programs such as *mpstat*, *vmstat*, *iostat*, *top* and *htop* which are included in several Linux distributions can read those performance counters behind the scenes and present statistical information on the behavior of the system. Core utilization, number of interrupts, and memory usage are some of statistics that can be returned. These programs can provide indications as to where the bottlenecks are.

For more detailed analysis, *oprofile* can offer more in-depth information, revealing CPU performance counters to the analyst. *Oprofile* can give detailed information, such as what functions/subroutines are consuming more CPU cycles or have a higher locking contention than others. In the latest kernel versions, another performance tool, *perf*, is included as part of the kernel [14].

For further analysis and highly accurate and complete performance numbers, platform-specific profiling tools, such as Intel® VTune™ tools, can be used. These tools are often more accurate, however, they may require a high degree of tool and hardware architecture experience in order to give accurate and valid performance results. Also, the tools may not be GPL licensed.

### Ongoing development in the kernel

The Linux kernel is “living” software; every day there are patches that impact different areas such as functionality, security, and performance. Care must be taken when deciding what kernel version to use. Small kernel versions or even small patch versions can have great impact on the performance of a Linux-based networking device.

# Experimental Numerical Data Based on IP Forwarding

### Objectives of the experiment

In this section we present the example of IP forwarding (routing) in the Linux kernel. We showcase the difference between the same system with no software optimizations applied and with all the possible optimizations we

could identify. It can be seen that the performance improvement was very significant.

## Test Equipment and Software

The platform was based on a single Intel® Xeon® processor for the throughput tests. Table 1 lists the system details and Figure 10 shows the setup.

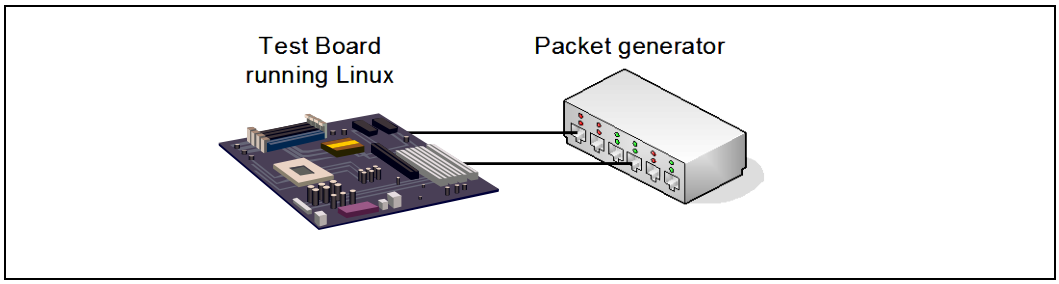
Several NICs were used:

- For 1 GbE, the Intel® 82576 Gigabit Ethernet Controller [15] was used.
- For 10GbE, the Intel® 82599 Gigabit Ethernet Controller [16] was used.

**Table 1. Test System Details**

| | |
|---|---|
| **Chipset** | Tylersburg 36D-B3 |
| **CPU** | Intel® Xeon® processor @ 2.53Ghz |
| **MEMORY** | 6144 MB DDR3 |

**Figure 10. Throughput Test Setup**




The chosen Linux distribution was Vyatta, a router-oriented Linux distribution [17]. Vyatta 4.1.4 VC was installed and a customized 2.6.28.1 SMP kernel was built. The kernel was focused on IPv4 and was built as lightweight as possible (that is, support for USB, sound, and IPv6 were compiled out of the kernel). For the 1GbE card, the igb 1.3.8-6 driver was built, where for the 10GbE cards, the ixgbe 2.0.24.4 was used. Both drivers were NAPI [18] enabled.

A packet generator was used. For 1GbE traffic, Spirent* SmartBits* was used where for 10GbE traffic, IXIA* XM12* with an LSM* 10GXM2XP-01 module was used. In both cases, the acceptable packet loss was 0.01%.

## Testing Methodology

The throughput tests executed were based on RFC 2544 [19]. Packets were injected into the first port of the network card and out from the second port

after having been IP forwarded by the Linux kernel. The packet generator was able to determine if there was any packet loss.

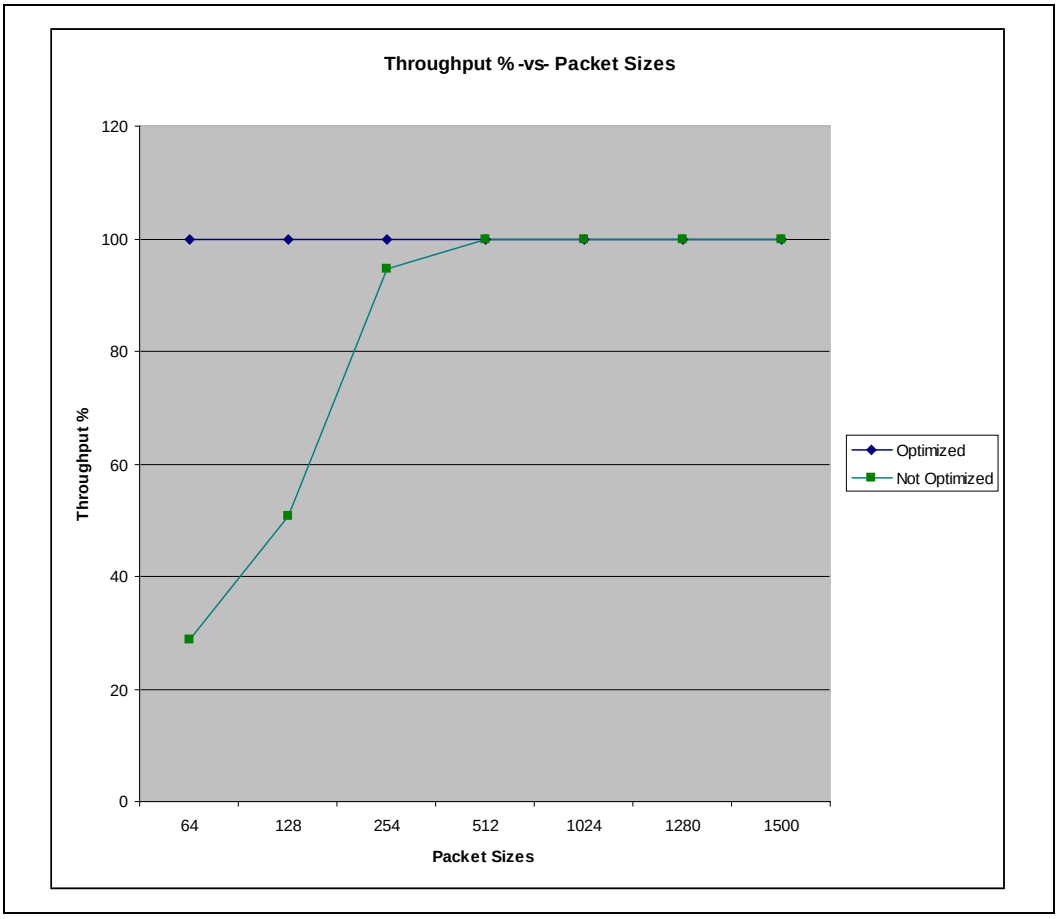
## Results

***Note:*** The term optimized or non-optimized in this case refers only to software. Some of the software optimizations include different BIOS settings, kernel with configuration that is only necessary for IPv4 networking, SMP affinities set for cache coherency, and driver brought up with the different parameters.

### 1 GbE Results

At 1 GbE speeds, the differences between an optimized versus a non-optimized setup can be seen (Figure 11). The non-optimized system achieves line rate at 512 byte packets and maintains it for bigger ones. On the other side, the optimized system achieved line rate from the smallest valid IP packet size at 64 bytes. It is worth noting that for the 64 byte packets, the non-optimized system only managed to forward approximately 30% of the possible throughput.

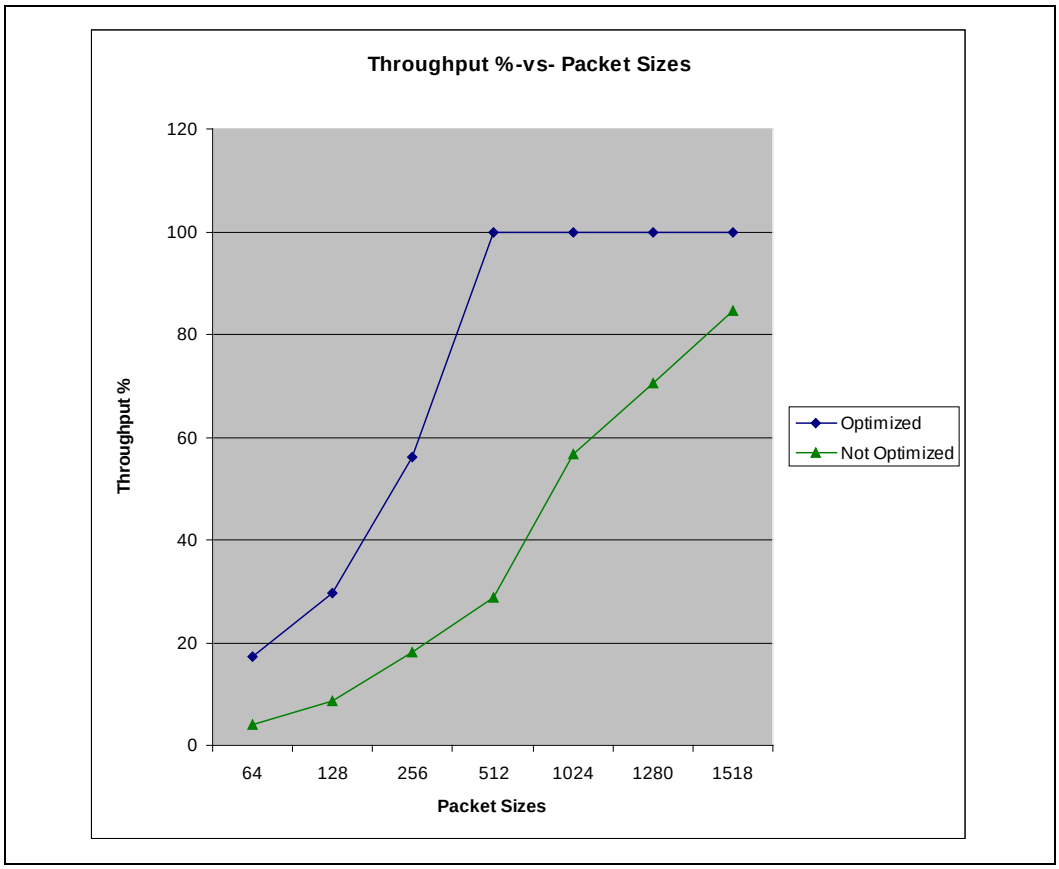
**Figure 11. 1 GbE IP forwarding in the 2.6.28 kernel**



### 10 GbE Results

At 10 GbE, things get more interesting because for small packet sizes the amount of packets per second that the system needs to cope with is ten-fold. It is clear in Figure 12 that a non-optimized system cannot achieve line rate even for 1512 byte packets. The optimized system achieves line rate at 512 byte size packets, at which size the non-optimized system achieves approximately 30% throughput.

**Figure 12. 10 GbE IP forwarding in the 2.6.28 kernel**




# Conclusion

The performance optimizations required in the Linux kernel are non-trivial because there are many parameters that can affect the system. Careful analysis must be performed, specifically, the data path needs to be analyzed both as a single entity and also as part of the whole system. Key factors for the performance of a networking application are:

- Hardware Architecture
- Linux Kernel Configuration

- Driver parameters
- Administrative Linux networking parameters

Some performance factors can be predicted and should be decided in the up-front design. Other factors are not visible until the prototyping stage; this is where a good set of tools as described in this paper can help identify and solve bottlenecks.

The Intel® Embedded Design Center provides qualified developers with web-based access to technical resources. Access Intel Confidential design materials, step-by step guidance, application reference solutions, training, Intel's tool loaner program, and connect with an e-help desk and the embedded community. Design Fast. Design Smart. Get started today. http://intel.com/embedded/edc.

# References

1. http://www.intel.com/technology/quickpath/introduction.pdf - An Introduction to the Intel® QuickPath Interconnect
2. http://www.kernel.org/pub/linux/kernel/people/christoph/pmig/numamemory.pdf - Local and Remote Memory: Memory in a Linux/NUMA System
3. Deri Luca, Fusco Francesco – Exploiting Community Multicore Systems for Network Traffic Analysis
4. http://www.intel.com/network/connectivity/vtc_ioat.htm
5. Karthikeyan Vaidyanathan, Dhabaleswar K. Panda – Benefits of I/O Acceleration Technology (I/OAT) in Clusters
6. http://www.intel.com/technology/quickdata/whitepapers/sw_guide_linux.pdf - Intel® QuickData Technology Software Guide for Linux*
7. Ram Huggahalli, Ravi Iyer, Scott Tetrick – Direct Cache Access for High Bandwidth Network I/O
8. http://download.intel.com/design/network/datashts/82576_Datasheet.pdf - Intel® 82576 Gigabit Ethernet Controller Datasheet
9. http://download.intel.com/design/network/datashts/82599_datasheet.pdf - Intel® 82599 10 GbE Controller Datasheet
10. Deborah T. Marr, Frank Binns, Devid L. Hill, Glenn Hinton, David A. Koufaty, J. Alan Miller, Michael Upton - Hyper-Threading Technology Architecture and Microarchitecture
11. http://www.ntop.org/PF_RING.html - PF_RING Homepage
12. IBM RedBooks - Eduardo Ciliendo, Takechika Kunimasa – Linux Performance And Tuning Guidelines
13. http://linux.die.net/man/8/ethtool Linux* manpage for the ethtool command

14. https://perf.wiki.kernel.org/index.php/Main_Page - Wiki page for Performance Counters for Linux*
15. http://download.intel.com/design/network/ProdBrf/320025.pdf – Product Brief for the Intel® 82576 Gigabit Ethernet Controller
16. http://download.intel.com/design/network/prodbrf/321731.pdf Product Brief for the Intel® 82599 Gigabit Ethernet Controller
17. http://www.vyatta.org/ – Vyatta Core* software official website
18. http://irtf.org/charter?gtype=rg&group=rrg Internet Research Task Force web page for Routing Research Group
19. http://www.ietf.org/rfc/rfc2544.txt – Benchmarking Methodology for Network Interconnect Devices RFC 2544
20. Jamal Hadi Salim, Robert Olsson, Alexey Kuznetsov - Beyond Softnet
21. Lawrence G. Roberts - The Next Generation of IP – Flow Routing, SSGRR 2003S International Conference, L'Aquila Italy

## Acronyms

| | |
|---|---|
| CPU | Central Processing Unit |
| DCA | Direct Cache Access |
| GbE | Gigabit Ethernet |
| I/OAT | Input/Output Acceleration Technology |
| IRQ | Interrupt ReQuest |
| MSI-X | Extended Message Signaled Interrupts |
| NAPI | New API |
| NIC | Network Interface Controller |
| NUMA | Non-Uniform Memory Access |
| PCIe | Peripheral Component Interconnect Express |
| QPI | Quick Path Interconnect |
| RFC | Request For Comments |
| RSC | Receive Side Coalescing |
| RSS | Receive Side Scaling |
| SMT | Simultaneous Multi-Threading |





§§