



Intel[®] Technology Journal

Communications Processing

Network Processing Performance Metrics for IA- and IXP-Based Systems

Network Processing Performance Metrics for IA- and IXP-Based Systems

Peter Brink, Intel Communications Group, Intel Corporation
Manohar Castelino, Intel Communications Group, Intel Corporation
David Meng, Intel Communications Group, Intel Corporation
Chetan Rawal, Intel Communications Group, Intel Corporation
Hari Tadepalli, Intel Communications Group, Intel Corporation

Index words: benchmarks, computer networks, IA32, Intel Architecture, IPSec, IPv4, IPv6, IXP, L3 Forwarding, MPLS, network processor, packet processing, PCI, performance analysis, VPN

ABSTRACT

The performance measurement and analysis of communications applications can be very challenging due to the diversity of applications, workloads, and operating hardware/software environments and the emerging state of standard benchmarks. In this scenario, performance analysis of the hardware and software sub-systems can provide a good basis for understanding and correcting performance bottlenecks and estimating the performance of newer platform-application combinations. Industry-standard benchmarks (such as SPECint), while validating processor and memory performance, do not address the complex interactions between the network, IO complex, CPU complex, and operating system in a packet-processing application. Since network processing encompasses diverse applications, the performance metrics and methodologies tend to be application-specific.

In this paper, we provide an overview of the methodology for determining and measuring various architectural parameters that impact the performance of a network processing system. Instead of the performance numbers of specific products, we focus on the overall methodology with selected examples: IPv4, IPv6, MPLS, L3 forwarding, Firewall, and VPN. The performance studies, based on Intel Architecture (IA, Intel's desktop processor) and IXP (Intel's network processor), detail the following aspects of performance analysis:

- Identification of architectural parameters of relevance to each networking application.
- Effective ways of measuring the performance based on those parameters.

- Relating the observed system performance to the architectural parameters.
- Translating the results of the analysis into opportunities and tradeoffs for improving the system.

INTRODUCTION

As the digital information age evolves to fulfill newer usage paradigms such as real-time commerce, ubiquitous connectivity, 3D gaming, video streaming, or instant messaging, the computing industry, and the platform architectures in particular, are under an ever-increasing demand to provide greater performance from all computing infrastructures: processors, IO, network, and storage.

A reliable way of characterizing the overall performance of a computing platform is to run industry-standard software benchmarks such as SYSmark, SPECint2000, SPECfp2000, 3DMark, TPC, and Quake [1]. These benchmarks load the platform with synthetic computational workloads representative of a wide variety of software applications that are required to be run in their target scenarios. Given their focus on desktop or server applications, these benchmarks emphasize CPU performance, memory bandwidth, and the impact of memory-to-CPU latencies. While providing a "signature description" of a platform's computing performance, they do not reflect the additional and somewhat orthogonal requirements of communications applications. Benchmarking the performance of networking platforms is further complicated by the absence of standardization around platforms, operating systems, applications, and the emerging state of benchmarks.

In this paper we describe the results of the ongoing efforts in Intel's Communications Infrastructure Group (CIG) towards developing a rigorous framework for understanding network processing performance. To best comprehend the architectural implications of network processing performance, our studies are based on two fundamentally different, but complementary processor architectures, from Intel: the IA32 processor family and the IXP networking processor family. While the system-level throughput and latency requirements are dependent purely on the networking applications under consideration, these requirements translate into different sets of architectural constraints in the two platform environments (IA and IXP) and hence need different methodologies.

In the case of IA32-based platforms, we describe a methodology for profiling the networking throughput of L3 forwarding, firewall, and VPN gateway applications and relate the observed performance limitations to the defining characteristics of the platform: CPU throughput, interrupt processing throughput, and IO to memory bandwidth.

The Intel IXP family of network processors is used to accelerate data plane processing to speeds typically in the range of several gigabits per second. Example applications of IXP processors are IP Routers and Multi-Protocol Label Switch (MPLS) switches. In the case of IXP-based network processors, we focus on performance metrics such as forwarding rate, latency, and the number of label switch paths as functions of the frame sizes and underlying hardware features of the network processor, such as the instruction set, and DRAM and SRAM bandwidth. We demonstrate performance improvements through a better utilization of the NPU architecture in software. Our emphasis on these hardware-level performance metrics, instead of on system-level performance metrics (such as those characterized in the three NPF benchmarks [6-9] IPv4, IPv6, and MPLS), is motivated by the need to characterize and predict the performance of the packet-processing subsystem (i.e., the network processor).

The rest of our paper is then organized as follows. In a processor agnostic setting, we differentiate the workload characteristics of network-processing applications from desktop or server applications. We state performance metrics at the hardware and system levels. We describe methodologies and results of measuring the system-level metrics on IA32 platforms with some specific applications: L3 forwarding, firewall, and VPN. The latter sections describe methodologies and results of measuring system-level metrics on IXP platforms.

In both IA32-based platforms and IXP-based platforms, our results and analysis are applicable in two complementary ways: performance improvements through software development targeted to the hardware architecture and hardware architecture improvements that suit networking applications.

NETWORK PROCESSING CHARACTERISTICS

Processing in the network is broadly classified into two categories: *data plane processing* and *control plane processing*, depending on the nature and extent of the processing that is required by the host CPU.

In data plane processing, the processor is involved with moving data from one external endpoint to another. The platform is mostly agnostic to the data transported. The nature of the work done by the CPU can be diverse and with complexity ranging from routing or forwarding (low CPU processing) to firewall, encryption, or virus scanning (heavy CPU processing). Meeting external line rates without losing packets makes it imperative that all platform latencies be kept to a minimum.

Control plane processing supports data plane processing by receiving packets over the network and updating the system state. Updating route lookup tables in IP forwarding and setting up an IPSec tunnel between two remote hosts are some examples of control plane processing. Compared to data plane processing, control plane processing involves a heavier CPU processing over a relatively smaller quantity of data.

Network processing is characterized in terms of "packets" with well-known size bounds. Every packet entering a communications platform can be viewed as undergoing a *packet processing* pipeline with four distinct stages:

1. *Receive Stage*. In this stage, a packet received over the external or fabric interface is reassembled and transferred to DRAM according to a packet descriptor that is either preloaded or created in-line in SRAM.
2. *Packet Processing Stage*. If the packet requires data plane processing, then various operations are performed in sequence: packet validation, IP lookup, TOS, TTL update, determining outgoing interface and MAC address, and optimal metering and classification.
3. *Quality of Service Stage*. This involves queue management and scheduling based on priority queues in SRAM.

4. *Transmit Stage.* The packet is retrieved, segmented, and transmitted by the framer over an external interface or the fabric.

The above stages of packet processing are generic to all platforms (e.g., IA and IXA) and to software and network protocol architectures (e.g., IPv4 and ATM). Implementation and hardware-software partitioning of each stage are architecture specific. For example, the (programmable) microengine-based architecture of IXP offers hardware support for all data plane processing functions with scope for assigning a separate microengine to each stage of the pipeline. In the case of IA32, both control plane and data plane processing are implemented in software under the support of an embedded operating system. In the case of IXP, all OS functions such as task switching, synchronization, and queue management are implemented in hardware.

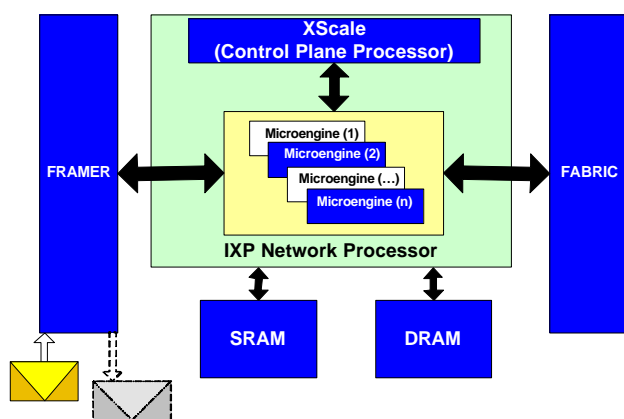


Figure 1: Packet processing pipeline

[Figure 1](#) illustrates the IXP processor architecture for implementing the above described pipeline.

The above description of a generic packet processing pipeline demonstrates many characteristics of networking applications that set them apart in many ways from desktop and server applications:

- *Embedded.* Networking applications are required to operate in stand-alone mode, requiring high reliability from the platform, OS, and applications.
- *Dedicated applications.* Networking platforms run a small number of standards-based applications while servers and desktops support a large variety of applications.
- *Data transience.* In a networking platform, a vast part of the data received over the network ports is transient with little processing or storage needs in the local processor. This short-lived nature of packets defines platform performance constraints that are more stringent than those required for

desktop and server applications. In contrast, most of desktop and server computing involves CPU-intensive processing and storage, suggesting a *data termination* characteristic.

- *Lack of data locality.* Desktop and server applications such as word processing, web browsing, graphics rendering, or web serving exhibit a large degree of locality of data that is critical to exploiting modern processor caches. Networking packets are short lived and need little processing by the CPU, hence larger and faster memory or cache may not translate into network throughput gains.
- *Criticality of IO bandwidth.* Since the route taken by a packet inside a networking platform involves an ingress device, system memory, and an egress device, it is critical that all interfaces between IO, memory, and CPU have adequate bandwidth to support the flow of packets and the associated control data (e.g., writing to the control registers of the NIC).
- *Real-time requirements.* Packet processing involves frequent CPU interrupts generated by network devices. To prevent data loss at high packet arrival rates, the interrupt-processing ability of the platform must scale with the packet transfer rate. A low packet latency also implies the ability of the platform and OS to respond in real time to packet arrivals. Most desktop and server applications, in contrast, are free from real-time considerations.
- *Privileged-mode execution.* Most networking applications are designed to run in the kernel mode to avoid context-switching overheads. Desktop and server operating systems need to support diverse applications; hence, they impose frequent switches between user and kernel modes.

CLASSIFICATION OF PERFORMANCE METRICS

Desktop and server benchmarks measure platform performance in terms of user ended “operations” per second. In networking applications, throughput in bits transferred per second and packet transfer latency in seconds are the primary metrics of interest. Besides these basic metrics, there may be other metrics of interest that depend on the application context e.g., *packet error rate*, *throughput under constraints* (rule bases for firewalls), *connections per second* (for VPN tunnels), and *number of connections* (for VPNs).

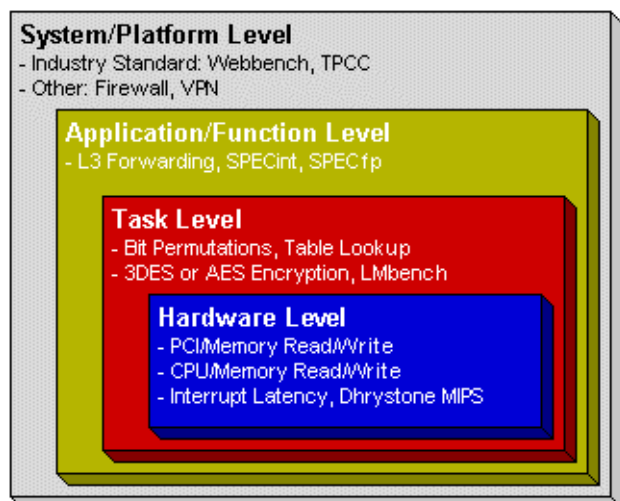


Figure 2: Classification of performance metrics

Based on the performance features of interest, Intel’s CIG proposed [7] a 4-level framework (Figure 2) for classifying potential networking benchmarks. In this model, a higher-level benchmark (e.g., L3 forwarding–Level 3) can be broken down into several software tasks (e.g., table lookups–Level 1, context switches–Level 2), each one best modeled by another benchmark at the hardware level.

Our measurements and analysis of networking examine one or more representative benchmarks at each level.

IA32 PERFORMANCE

Figure 3 illustrates the “standard” platform that we used for testing in the IA32 portion of this paper. The details of this platform are as follows:

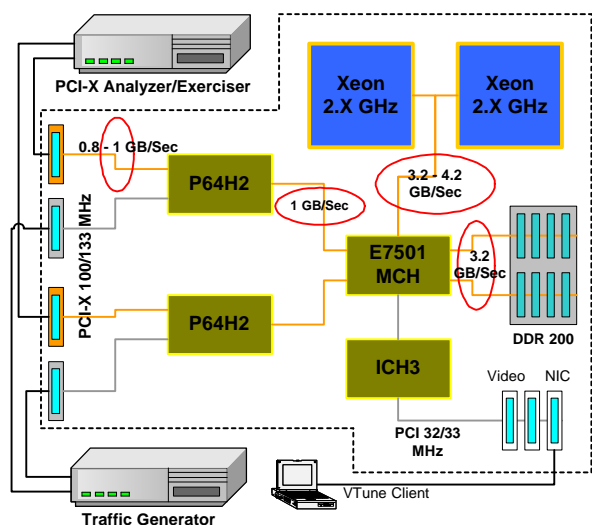


Figure 3: Standard IA32 network test platform

- *Hardware.* Dual Intel® Xeon™ processors, Intel E7501 chipset, two P64H2 IO bridges, and two channels of DDR200 as a main memory. The PCI-X exerciser connects directly to the P64H2 through PCI-X, and the traffic generator connects through 2 82545-based Gigabit Ethernet cards.
- *OS/Software.* Linux* (Kernel version 2.4.18) using the FreeS/WAN 1.99 release VPN software.

HARDWARE–IO TO MEMORY BANDWIDTH

The “Hardware Level” benchmarks form the lowest level of the IO performance benchmark hierarchy. They are intended to characterize how well a platform performs elementary hardware-related operations such as PCI-X memory read, memory write, etc. On a typical IA32-based networking platform, a small-size packet forwarding traffic results in many memory accesses; therefore, knowing how well a platform can perform these hardware-level memory operations, can give a good indication of the upper bound of the platform performance without any software overhead. The hardware-level performance in terms of the *Throughput*, *Bus Utilization* and *Efficiency* can be measured under different conditions.

The PCI-X/memory performance is measured for the block READ and WRITE operations for the burst sizes from 64 Bytes to 4 Kbytes. The measurements are done with a single PCI-X agent, multiple PCI-X agents on the same IO bridge, and multiple PCI-X agents on multiple IO bridges. The performance is measured for various combinations of READ and WRITE operations. Further, these measurements are taken while the CPU is idle, as well as with the concurrent CPU load. Thus, two PCI-X agents and the CPU doing various combinations of READs and WRITEs can generate many different scenarios.

The test scenarios described above can give a very good indication of the performance of the IO Bridge, performance of the interface between the memory controller hub and the IO controller (also known as “hublink”), and the streaming capability of the Memory Control Hub (MCH). The best-case PCI-X performance can be obtained by “Single PCI-X agent with CPU idle.” The performance limitations of hublink can be found by

® Intel Xeon is a trademark of Intel Corporation or its subsidiaries in the United States and other countries.

*Other brands and names are the property of their respective owners.

using the “Multiple PCI-X agents on the same bridge” cases. These cases are useful in finding out the performance limitations resulting from the arbitration scheme and the buffering capabilities of the MCH. In addition to these cases, the “Concurrent CPU load” cases can be useful in measuring the performance of the CPU/Memory path and stressing the MCH arbitration further.

Out of all these scenarios, the most basic case, a single PCI-X agent performing READs while the CPU is idle, is discussed in this paper.

PCI-X Performance Parameters

PCI-X bus *Throughput*, which is the number of data bytes per unit time, is the term most often used to specify bus performance.

$$\text{Throughput} = \frac{\text{Bytes transferred}}{\text{Time}} \quad \text{MB/Sec}$$

Also,

$$\text{PCI-X throughput (MB/sec)} = \text{PCI-X Efficiency (\%)} * \text{PCI-X Bus Utilization (\%)} * \text{Bus Speed MHz} * \text{Bus Width (bytes)}$$

Another important parameter is PCI-X *Utilization*, which is defined as the ratio between the time used by the PCI-X transactions and the total measurement time.

$$\text{Bus Utilization (\%)} = \frac{\text{Busy Clocks}}{\text{Total Clocks}} \times 100$$

If every clock is used by one of the agents on the bus, bus utilization would be 100%. If the PCI agent is not capable of using the bus to full capacity, bus utilization would be less, and that may be the performance bottleneck.

The third parameter is PCI-X *Efficiency*, which is the ratio between the bus time used in transferring the data portion of the transactions and the total time used by agents for completing those transactions.

$$\text{PCI-X Efficiency (\%)} = \frac{\text{Data phase clocks}}{(\text{Data phase clocks} + \text{Non-data phase clocks})} \times 100$$

Single PCI-X Agent Memory READ with CPU Idle

In this case, the performance of the PCI to Memory path is measured while the CPU is running only minimum tasks such as servicing the OS routines, etc. The Single PCI-X agent tests measure performance of IO bridges, the streaming capability of the North Bridge (MCH), and the interconnecting link (hublink) between the IO Bridge and the MCH. For maximum bus utilization, an agent is configured to generate more than one transaction without waiting for the completion of the previous transactions.

Measurement

Due to PCI-X read latency in the chipsets, all PCI-X memory read operations result in split transactions or retries. The efficient use of the time between memory read request and its response improves the efficiency. By issuing multiple transactions, a PCI-X agent can pipeline the transactions.

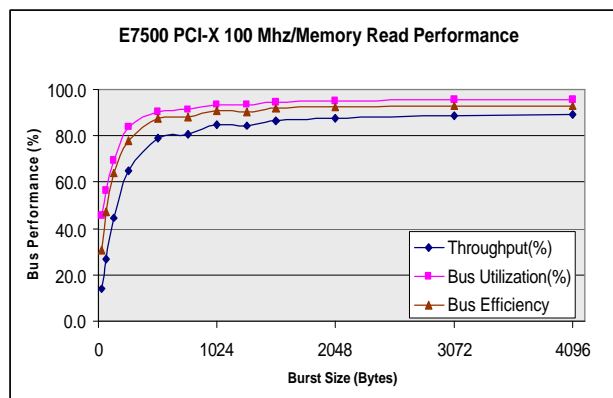


Figure 4: PCI-X Memory read block tests

Figure 4 shows results of PCI-X 100MHz/64-bit Memory Read Block tests.

Analysis

Each memory Read block transaction results in Split response. Up to a burst size of 512 bytes, the average Split rate is 100%. This means that every split transaction is completed by a single split completion. Above 512 bytes, P64H2 arbitrates among pending split completions after every 512 byte transfer, i.e., 64 clocks, the PCI-X Maximum Latency Timer value.

Each memory Read block transaction results in a Split response. This operation takes 5 PCI-X clocks. After some latency, there are one or more split completion cycles. Each split completion there is an overhead of 4 bytes consisting of address, attribute, and response phase and one wait state that is introduced by the Exerciser card.

For a burst size of 256 and less, bus utilization is poor, which results in a lower throughput number.

Optimization

Since PCI-X throughput is proportional to both PCI-X utilization and PCI-X efficiency, understanding these actors can help in designing a product with optimized performance for a specific application.

PCI-X Device

Sometimes bus utilization is limited by the architecture and design of the PCI-X IO cards. To illustrate this, the PCI-X Exerciser card used in this experiment requires a

finite number of clocks after completion of one transfer to generate the next transaction. This results in idle clocks on the PCI-X bus. Some PCI-X agents do not support multiple transactions. All these factors may result in reduced bus utilization.

A limited buffer size of the PCI-X agents can break a large transaction into smaller parts. Also, some PCI-X IO cards break large transactions in multiple parts to allow other agents to access the PCI-X bus. All of these factors may result in degraded efficiency.

Chipsets

The chipset itself may introduce some idle clocks between two transactions for the bus turnaround. Also the IO Bridge may take several cycles to grant bus access to an agent. In modern IA32 chipsets, most of the PCI-X memory read commands result in two split transactions, which introduces overhead of at least 4 clocks per split completion. All these factors result in lower efficiency.

If there are multiple agents on the same PCI-X bus, the IO Bridge arbitrates among the transactions, which may result in multiple transfers. Further, if the transactions are in both directions (e.g., WRITE and READ); they are arbitrated on both the hublink and PCI-X bus. The MCH also arbitrates among different hublinks and Front-Side Bus (FSB) traffic. Heavy traffic on other links may result in multiple transfers for a transaction on a PCI-X bus.

PCI-X Architecture

Any PCI-X transaction has at least three clocks of overhead because of the address phase, the attribute phase, and the response phase; therefore, PCI-X efficiency can never be 100%. Sometimes the PCI-X master cannot launch a new transaction because it already issued the maximum possible transactions and it has to wait for at least one of its outstanding transactions to complete. This kind of situation arises particularly at the lower burst sizes or when multiple agents on different buses request more transactions than the system can serve.

If the PCI-X/memory transaction start address is not aligned to a cache line boundary, the transaction may be broken at the next Aligned Data Boundary (ADB) by the system or by the master itself.

The hardware-level performance measurement gives a good indication of the upper bound of platform performance, without any software overhead. Once the performance characteristics of the hardware are known, we need to understand the performance bottlenecks with the software running in the appropriate operating environment.

APPLICATION–L3 FORWARDING

Layer 3 (L3) Forwarding is a network function that looks only at low-level information in a network packet to determine where next to send that packet. It uses the Layer 3 information in the packet (the IP information, as in the IP Address). There are various levels of forwarding, from the simplest case where no look-up is performed and the packet is immediately forwarded, to a look-up being performed, to adding extra processing to the operation using a function such as a firewall, all of which are characterized by both throughput and latency. For most systems, throughput is by far the most important. In our example, we use an IA-based system during the forwarding while taking an external viewpoint.

Network Topology (the 1:1, and N:N cases as mentioned above) and Packet Distribution (varying packet size from 64 to 1518 bytes per packet) are types of parameters we can modify during the measurements to characterize throughput as a function of traffic type and network configuration.

Outputs from these measurements include the following:

- Data that indicate how good this system is at performing forwarding, which is useful as a benchmark when comparing our system to other systems.
- Measurements that can provide enough insight to identify functional bottlenecks in the system.
- Identification of any low-level primitives that could benefit from silicon, software acceleration, or other optimization methods.

Methodology

To keep the number of system configuration variables to a minimum, we need to identify which parameters are useful for this measurement. The following are types of parameters we can modify during our measurement and analysis:

- *Processor and System* (single or dual processor system, processor frequency, and network card).
- *OS* (kernel version, interrupt assignment to a specific CPU, optimizations).

We have limited the number of system configuration items for this test to keep the number of tests to a manageable level. For instance, we do know that the performance scales linearly with processor frequency, though not with a 1:1 ratio. With those factors in mind, we connected a traffic generator to measure the effective throughput of the system while varying the packet size.

Measurement

For the measurements, we measured the maximum throughput at 0% packet loss while varying packet size and using the following configurations:

- 1:1 (no route lookup) and 128:128 (8:8 networks with 16 nodes per network).
- A firewall with 200 and 512 rules.

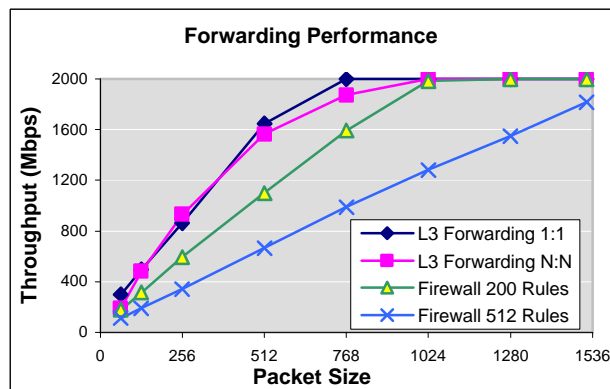


Figure 5: Single processor forwarding

Analysis/Optimization

The first test that we ran amounted to comparing the various forwarding cases (1:1, N:N, firewall 200 rules, and firewall 512 rules) on a single processor system; attempting to prove that adding extra processing overhead causes a reduction in throughput. As the chart displays, there is little to no difference between 1:1 and N:N, but as we added more processing with each packet, the 200 and 512 rule cases, it adversely affected throughput.

From the measurement data, it is clear that the forwarding function is not limited by the processor, but as the load increases as in the 512 rule case, the system is incapable of reaching the 2 Gbps line rate. The next step then becomes an analysis of how to improve the baseline function as that improves the generic forwarding capability of the system.

During our analysis of the situation, we determined several different methods for improving the system. At the culmination, though, the main factors ended up being the following:

- *Processor synchronization.* In a dual-processor system, one processor must lock variables, in this case the routing tables, in memory to prevent the other processor from making any modifications until the first processor is done with it. This process is called **serialization**, because the variable access

forces two or more processors to act in a serial fashion instead of in parallel.

- *Interrupt distribution.* The additional Interrupt Requests (IRQ) and scheduling overhead contribute to performance degradation at lower packet sizes. The question then becomes, “Why?”

We made these main observations with a standard 2.4.18 (default) kernel:

- The OS must write the Task Priority Register to the MCH when a task’s priority changes, something Linux *does not do*.

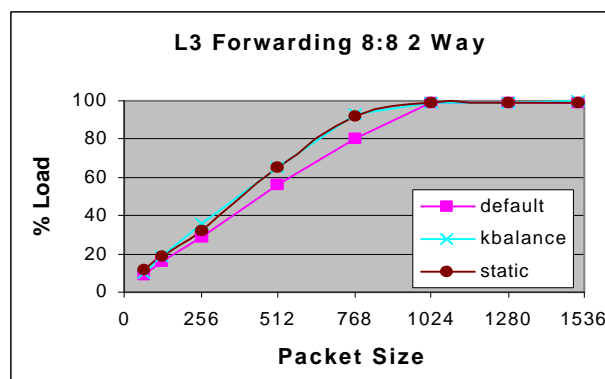


Figure 6: Dual processor forwarding

- The MCH directs interrupts based on the CPU ID or preferably the CPU running the lowest priority task, *but the task priorities never change!* A chipset-based tiebreaker chooses the lowest CPU ID (CPU0). Thus, 80% of interrupts go to CPU0, 16% go to CPU1, etc.

There are various patches available to correct this problem:

- The first patch automatically points the interrupt controller to the next processor for an Interrupt Service Routine (ISR). You lose some cache locality this way as the same processor does not process consecutive interrupts (not that important for transient data), though you do get better distribution. This did not amount to much, if any, improvement over the default kernel.
- A user can choose static device assignment, assigning all interrupts for a particular interrupt source to a specific CPU (static in [Figure 6](#)). This really works best when you have an equivalent number of sources as you have CPUs and an even interrupt load.
- There are also a number of software patches that use things like CPU utilization (load balancing) and the

number of IRQs per CPU (Kbalance) to decide where to assign an interrupt.

Both the Kbalance and the static IRQ patches did make a significant difference in system performance, but for L3 forwarding on Linux, the throughput is significantly lower than what was observed on the hardware capability. Our conclusion from this is that IO bandwidth is not the limiting factor; instead, OS overhead is.

SYSTEM LEVEL: VPN PERFORMANCE AND ANALYSIS

A Virtual Private Network (VPN) is a “tunnel” over a shared or public network infrastructure, where the data are encrypted and encapsulated at the source, decapsulated and decrypted at the destination, and routed by all intermediate nodes in the Internet. In a typical VPN setup, two private domains route packets between each other through VPN gateways situated on the respective edges, while hosts exchange unencrypted IP packets inside the private networks.

IPSec, adopted by the IETF, is the most popular IP layer framework for VPN gateways. For detailed overviews of VPNs and IPSec, we refer the reader to [2], [3], and [4].

In the next section we describe a methodology for measuring two key performance metrics for a VPN gateway running the IPSec protocol: networking throughput at different IP packet sizes; and a detailed breakdown of the CPU utilization among the software constituents of the IPSec stack. A related goal is to conduct a “hotspot analysis” by identifying functions with a high clock ticks per instruction (CPI). CPI is a key processor utilization metric: functions that utilize the CPU resources best tend to exhibit a CPI of less than 2, while functions with a CPI larger than 4 are suspect in that they lose instruction throughput owing to factors such as poor cache utilization, processor stalls, and excessive inter-processor synchronization. In the analysis section, we show how the CPU utilization data can be used to gain some valuable insights into the behavior of the software vis-à-vis the platform and the application (IPSec in this case).

Our measurements and analysis are based on the Intel Xeon platform described in the section on the [IA32 Performance](#) and the open source implementation of IPSec based on the Linux OS and the [FreeS/wan 1.99](#).

Methodology

[Figure 7](#) illustrates our experimental setup for IPSec encapsulation with 3DES (encryption) and MD5 (message authentication) as the tunneling options.

A network throughput exerciser (NTE) “pumps” IPv4 packets to the VPN gateway under test at speeds of up to 1 Gbps. VPN encapsulates these packets according to a preset IPSec tunnel descriptor and forwards the (encrypted) packets to the NTE. When the direction of the flow is reversed in the NTE, the same setup can measure the decapsulation performance. VPNs should not “choke” the flow coming from the VPN and hence should have a better IPSec performance than that of the VPN.

In addition, the VPN is connected to a CPU utilization tool (the Intel® VTune™ Performance Analyzer) run from a standard Windows® PC platform over a 10/100 Ethernet connection.

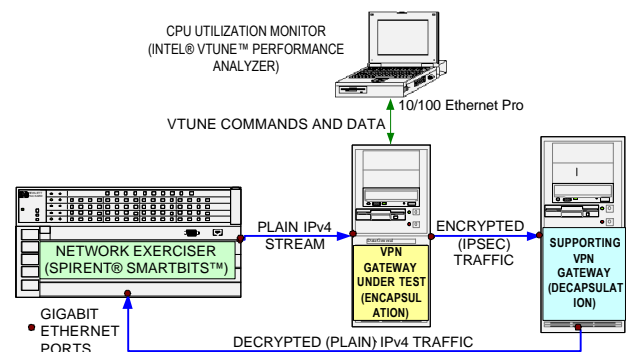


Figure 7: Setup for measuring throughput and CPU utilization for IPSec encapsulation

Results

For the VPN configuration described in the previous section, the maximum achievable throughput (in Megabits per second) and CPU utilization are profiled with the following parametric variations:

- *Default interrupt routing and static interrupt routing.* This is explained in the [L3 Forwarding](#) section of this paper.
- *Encapsulation and decapsulation.* We measure unidirectional throughputs separately and profile the corresponding CPU utilization.
- *Packet sizes.* These are varied progressively between 64 and 1518 bytes (the maximum size of an Ethernet frame).

The results are organized into two categories: system-level analysis that identifies performance issues related to the visible system configuration, and hotspot analysis that

® Intel VTune is a trademark of Intel Corporation or its subsidiaries in the United States and other countries.

“zooms” into the underlying software components and identifies the parts with poor CPU utilization.

System-Level Analysis

Figure 8 shows the variation of throughput with packet sizes for three different options of encryption and authentication: both 3DES and MD5 enabled, only 3DES enabled, and only MD5 enabled. We omit similar graphs for decapsulation and bidirectional throughputs and discuss the results.

- Throughput at higher packet sizes is at least 2.5 times the throughput at small packet sizes.
- Decapsulation throughput is up to 5% better than encryption for all packet sizes.

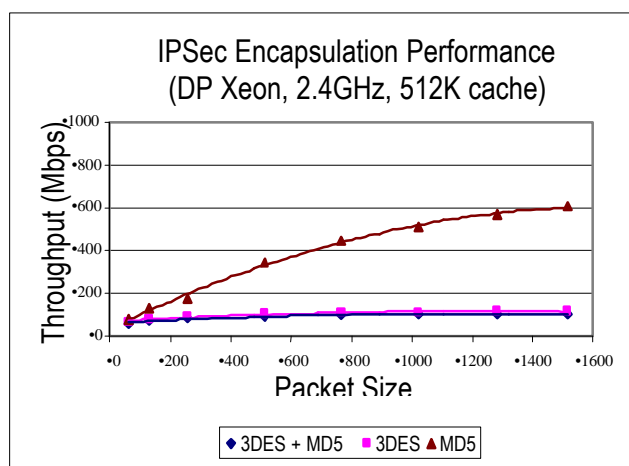


Figure 8: IPSec encapsulation performance vs. packet size for different 3DES/MD5 options

- The dual-processor (DP) kernel outperforms the single or uniprocessor (UP) kernel by approximately 25% in throughput. Although the workload is balanced across both CPUs, the overheads of interrupts, OS, and memory contention in the Symmetric Multiprocessing (SMP) scenario compromise the benefits of having a second processor.
- Adding 3DES over MD5 depresses the throughput by 80%, suggesting the intensely compute-bound nature of the 3DES algorithm.

Hotspot Analysis

Figure 9 shows CPU utilization for encapsulation at 64 and 1518 packet sizes for the DP Xeon platform. The results are discussed following Figure 9.

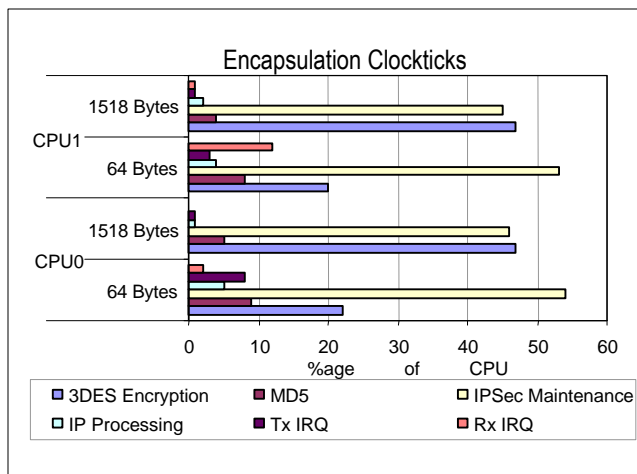


Figure 9: CPU utilization distribution among various functions in IPSec

- *Processing is well balanced across the two CPUs.* The CPU workload in IPSec can be analyzed in terms of three types of functions: encryption related, IPSec maintenance, and OS overheads.
- *Interrupt and OS overheads.* With larger numbers of packets at low packet sizes, the interrupt and OS overheads are significant at low packet sizes and negligible at large packet sizes.
- *Encryption-related functions (3DES and MD5).* 3DES accounts for a reasonable (22-29%) part of CPU utilization at small packet sizes and is the most dominant part (47 - 88%) at higher packet sizes.
- *Time spent on MD5.* The time decreases from 9% to 4% as the packet size increases from 64 bytes to 1518 bytes. Since the time spent in MD5 is proportional to the amount of data processed, this percentage is expected to be more at higher packet sizes. Since MD5 is not computationally as intensive as 3DES, the processing overhead per packet dominates the MD5 part of packet processing.
- *Efficiency of CPU utilization.* In encapsulation, 3DES and MD5 exhibit good CPIs of 1.3 and 2.08, respectively. In decapsulation, 3DES has a poor CPI of 5, suggesting poor implementation with many data dependencies.
- *IPSec maintenance.* This exhibits a similar trend as MD5. Its share of CPU load decreases from 50% to negligible as packet size increases from 64 bytes to 1518 bytes.

Recommendations for Optimization

The hotspot analysis outlined above reveals a few insights into the Linux 2.4.18 and FreeSwan-based implementation of IPSec:

- *Performance penalty arising from SMP synchronization overheads.* The high CPU utilization of IPSec maintenance code is also accompanied by a very detrimental CPI (as high as 23) in some key functions. The source code of these functions shows that they are heavy in spin locks, an SMP synchronization feature that can result in a severe performance penalty, when used inappropriately. Further, these spin locks are acquired and released mostly for reading, and seldom for writing, some critical memory sections. A suggested performance improvement is to include one Read lock and one Write lock around these critical sections. The Read lock can be a counted semaphore, thus allowing multiple readers, while any writing function needs to acquire all the Read locks and the Write lock before writing.
- *Excessive processor utilization by the cryptographic functions.* The high CPU utilization and a low CPI of the 3DES and MD5 functions suggest two performance tips: rewrite these functions so that the software targets the Intel Pentium® 4 processor architecture more closely and provide hardware (offload) support for cryptographic functions.
- *Per-packet overhead at lower packet sizes.* At lower packet sizes, the throughput is limited by the amount of processing needed to process a large number of packets rather than the amount of data. Any effort to improve interrupt handling latency or to serialize all the processing related to each packet (without having to reload the same packet from memory into cache multiple times) should improve the performance.

IXP PERFORMANCE

Hardware

In order to characterize IXP2400 performance as per the NPF IP, IPv4, and MPLS benchmark specifications, a dual-processor IXP2400-based hardware platform was developed, consisting of the following:

- Two IXP2400 600 MHz network processors each having 512 MB of 150 MHz DDR DRAM and two channels of 4 MB 200 MHz QDR II SRAM.
- Two IXD2410 media cards, each having a 133 MHz IXF1104 with 4 Gigabit ports.

Each IXP2400 has 2400 MBps of DRAM bandwidth and 1600 MBps of SRAM bandwidth available for packet

® Pentium 4 is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

processing. The system was designed to handle a maximum of 8 Gbps of Ethernet traffic or 4 Gbps per IXP2400. In the case of Ethernet packets, in the worst case, the system should be capable of handling a packet every 100 IXP2400 microengine cycles.

APPLICATION: IP PROCESSING

On the IXP2400-based platform, two NPF benchmarks were executed. The NPF IPv4 benchmark [8] and the IPv6 portion of the NPF IP benchmark [9]. The basic software architecture for both was the same and is shown in [Figure 10](#). The system was built using standard software building blocks available in the Intel IXA SDK microblock library.

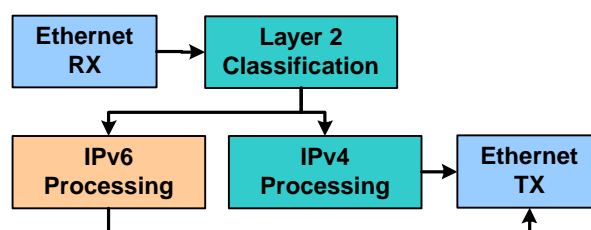


Figure 10: IP benchmark software architecture

All traffic for which the routes are present in the route table and for which ARP entries have been resolved is handled in the “data plane” by the microengines. Traffic that is destined for the higher-layer protocols on this system (i.e., route updates, ARP requests, responses, and IP option packets, etc.) is sent by the microengines to the Intel XScale® microarchitecture-based control processor.¹

In the data plane, the microengines receive packets from the data interfaces and store them in DRAM; the Layer 2 classification blocks fetch the packet header from DRAM, strip the Ethernet header, and classify the packet as IPv4 or IPv6 before sending it to the right IP processing block. The IP processing blocks validate the headers and then perform the software-based longest prefix match algorithm (LPM) IP lookup using a trie table² stored in SRAM. This is followed by Ethernet encapsulation and transmission to the data interface by the Ethernet transmit blocks.

The performance of this system is affected by the following factors:

- Size and prefix length distribution of the route table and distribution of the destination IP addresses of the

® XScale is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries.

¹ To be abbreviated as “XScale® processor,” henceforth.

² A Trie is a tree in which each parent node has three children.

incoming packets determine the number of lookups needed per packet (i.e., SRAM utilization).

- Size and rate of incoming packets. Packet size determines DRAM utilization, inter-arrival time, and the amount of processing performed, i.e., SRAM and microengine utilization.
- Percentage of packets sent to the XScale processor and the degree of processing needed per packet on the control plane.
- SRAM and DRAM utilization by the XScale processor for performing tasks such as table maintenance.

Methodology

The NPF Benchmark Implementation agreements clearly specify the methodology that needs to be used when performing network processor benchmarks. This includes the route tables to be used (28,895 Entry Mae-West snapshot for IPv4 and extrapolated 1,200 entry AS4554 snapshot for IPv6), the performance metrics to be measured and their presentation, as well as the parameters that need to be varied.

An IXIA traffic generator was used to generate 1000 unique packets per port (i.e., unique destination addresses) in the case of IPv4 and 100 packets per port in the case of IPv6 as shown in Figure 11. The destination addresses have the same overall prefix length distribution per port as that of the route table, leading to a symmetric traffic processing load per port. This ensures that the performance obtained reflects system constraints rather than test limitations.

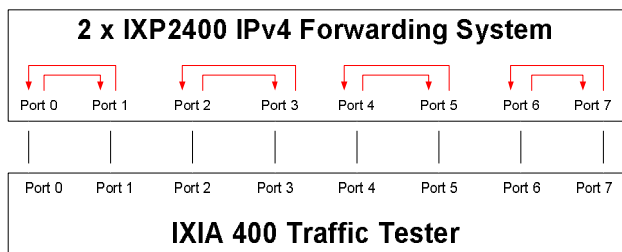


Figure 11: Traffic configuration

Measurement

In the case of IPv4 and IPv6, the following performance metrics are measured, with the Ethernet packet sizes being 64 (IPv4), 78 (IPv6), 128, 256, 512, 1024, 1518, and the Internet Mix (56% 64 byte, 20% 576 byte and 24% 1518 byte packets), which is typical of packet distribution for Internet traffic.

- The throughput latency (100%, 75%, 50%, 25%) of the throughput rate.

- The maximum number of route updates on the control plane when there is no traffic on the data plane.
- The forwarding rate when the performed route update is at 100%, 75%, 50%, and 25% of the maximum update rate.

The forwarding rate is chosen from one of the three options: (1) 100% data plane traffic; (2) 95% data plane traffic and 5% control plane traffic sent to the XScale processor and dropped; and (3) 99.9% data plane traffic and 0.1% Record Route IP option packets sent to the XScale processor and sent back down to the data plane for processing.

Analysis

The system was able to forward traffic at the line rate for all packet sizes in all three cases mentioned before. This implies that the system has enough processing power as well as memory bandwidth to be able to handle data plane traffic as well as a normal level of control plane traffic without packet loss.

There was no significant increase in latency when the rate was increased from 25% to 100% of the line rate. This also means that no component of the system is overloaded: hence, there is no degradation in performance. However, the 64 and 78 byte packets have a higher latency than the 128 byte packets as shown in Figure 12. This reflects on the system receive and transmit configurations that were set to allocate fetch and transmit 128 bytes blocks of data from and to the data interfaces.

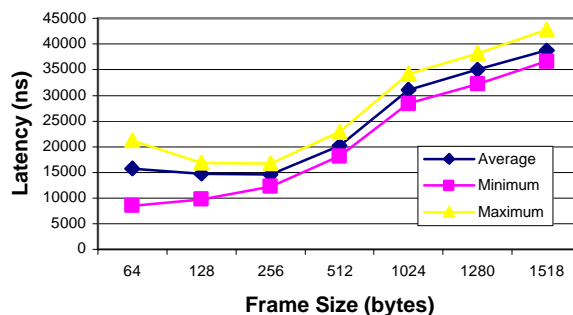


Figure 12: IPv4 Latency at 100% throughput rate

The system was capable of performing 96,292 IPv4 and 18,000 IPv6 route updates per second. This rate is also the rate at which the XScale processor places the maximum load on the memory.

A slight drop in forwarding rates was observed for the maximum possible route updates for minimum-size packets (64 byte IPv4 and 78 byte IPv6). However, the packet loss decreases significantly as the route update

rate is reduced. There is no packet loss seen for other packet sizes as shown in [Figure 13](#).

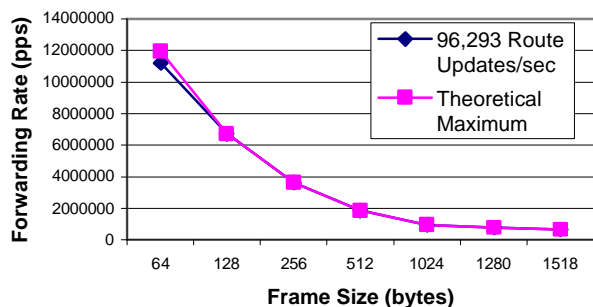


Figure 13: Forwarding rate with concurrent route updates

For minimum packet size, the microengines have very high SRAM utilization and the XScale needs to modify the SRAM-based trie tables. The XScale processor is configured to have significantly lower memory access priority than the microengines; however, when it gets access to the memory it locks the SRAM controller for significant periods to ensure data integrity. For the Internet packet mix (more realistic condition) we see no such impact.

APPLICATION: MPLS PROCESSING

The Multiprotocol Label Switching (MPLS) forwarding system is designed in order to characterize MPLS forwarding performance for the IXP network processors. The system architecture is shown in [Figure 14](#). The support hardware configuration is illustrated in Figure 11. In order to support MPLS functionalities required by the NPF benchmark specification, eight processing blocks are established in the system. The major MPLS processing blocks are label PUSH, label SWAP, and label POP. The details of these MPLS operations and the MPLS principle can be found in [11], [12], and [13]. The MPLS table lookup block is used to find a MPLS operation from MPLS table entries according to a given label, which is embedded in the MPLS packet. Other blocks including Ethernet RX, TX, IPv4 and Layer 2 (L2) classification are from the IXA SDK microblock library while the L2 classification block is modified to handle MPLS packets.

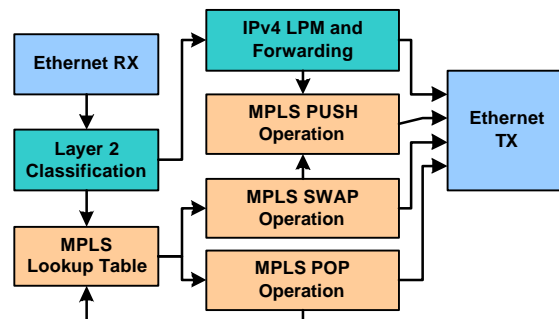


Figure 14: MPLS processing system architecture

In the MPLS forwarding performance measurement, the Ethernet streams are sent to the L2 classification block by the Ethernet RX block. The L2 classification block extracts network packets from the streams and dispatches the packets to different processing blocks depending on the packet type.

For the IP packets, the processing path is IPv4 → MPLS PUSH → Ethernet TX. This processing path characterizes ingress label edge router's (LER) performance, because an IP packet is converted to an MPLS packet. MPLS PUSH may add multiple labels to the packet label stack.

For the MPLS packets, there are multiple processing paths. The first path is MPLS table lookup → MPLS SWAP → Ethernet TX. This path characterizes transit label switch router's (LSR) performance, because the label in the packet is replaced by the label from an MPLS table entry. The packet type is not changed by this operation.

The second path is MPLS table lookup → MPLS POP → Ethernet TX. In this case, multiple labels may be removed from the packet's label stack. This path characterizes egress LER's performance, because an MPLS packet may be converted to an IP packet when all labels in the packet are removed out.

The last path is MPLS label lookup → MPLS SWAP → MPLS PUSH → Ethernet TX. This is a SWAP and PUSH operation. This characterizes a transit LSR's performance.

The worst case is the PUSH three labels operation in terms of performance. Each label is 32 bits, and three labels expand the packet length by 96 bits. That means more data is stored in memory and transmitted to the wire. This affects both forwarding speed and latency number.

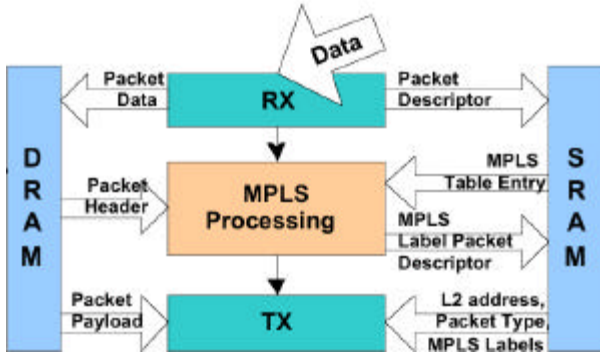


Figure 15: MPLS processing data flow

Data Flow in MPLS Processing

The major packet latency in MPLS forwarding results from the data movement between the microengine and off-chip memory. In order to reduce the latency, it is desirable for the MPLS forwarding system to minimize memory accesses while preserving complete functionality. The system data flow, which reflects this design approach, is shown in Figure 15. The packet data is stored in the DRAM while the packet descriptor is stored in the SRAM in the RX block. The DRAM is high-volume, inexpensive memory, but is relatively slow. It is, therefore, suitable to store packet data. The SRAM is low-volume, fast memory, but is expensive. It, therefore, is suitable to store the packet descriptor, which is updated by multiple microengines. The packet header is cached in the local memory for fast processing. MPLS processing is done on the packet header. After processing, the updated MPLS packet header is stored in the SRAM instead of DRAM in order to reduce memory accesses. TX reads L2 information and the MPLS header from SRAM, packet payload from DRAM, and transmits the whole packet to the wire.

Methodology

The methodology and system configuration to be used in MPLS processing is detailed in the NPF MPLS Benchmark implementation agreement [6] and is very similar to the previously discussed IP benchmark.

Measurement

The NPF MPLS Forwarding Application Level Benchmark specification defines Ingress Traffic, Transit Traffic, and Egress Traffic.

The parameters used are as follows:

- *IP packet sizes (octets):* 40, 110, 238, 494, 1006, 1264 and 1500.
- *Mae-West routing tables:* same as those in IPv4 measurements.

- *MPLS operations:* PUSH 1, 2, and 3 labels; SWAP/PUSH: 1 SWAP, 1 SWAP & 1 PUSH, 1 SWAP & 2 PUSH; POP: 1, 2, and 3 labels.

The metrics measured are forwarding rate, throughput, latency, and maximum label switch paths (LSPs) supported at the throughput rate.

An example of the benchmark test result for ingress LER forwarding MPLS packet at line rate for all packet sizes and for 1, 2, 3 PUSH operations is shown in Figure 16.

Analysis and Optimization

Memory access and instruction cycles are the major factors impacting system performance. The RX block is optimized for different packet sizes and minimum memory access.

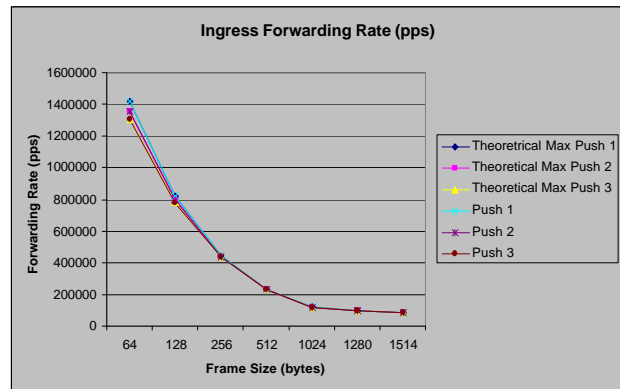


Figure 16: Forwarding rate for all packet sizes in ingress LER test

The MPLS Incoming Label Map (ILM) and the Next Hop Label Forwarding (NHLFE) tables are combined as one table. The parameters in an entry are arranged in such a way that only the first two long words from a selected table entry are read from SRAM at the beginning. The lookup algorithm then checks the entry operation. If no further parameters are required, the lookup is done and no more SRAM read operations are performed. If the operation needs more parameters from the entry, the remaining parameters are read. In 60% of the cases, only the first two long words in the entry are needed thereby reducing SRAM bandwidth usage by 25%.

Another optimization was to use SRAM to store new pushed or swapped MPLS labels instead of writing all packet headers back to the DRAM.

The transmit code was optimized to read Ethernet MAC addresses and MPLS label(s) from SRAM and place them directly into the transmit buffer (TBUF) element prepend section, thereby sparing a write to DRAM, which reduces the amount of DRAM bandwidth utilized.

In general, the system tuning focused on reducing instruction cycles, DRAM and SRAM accesses, and processing methods by taking advantage of the programmability of the IXP network processors.

CONCLUSION

Classically, desktop and server applications occupied the center stage for performance analysis in high-performance computing. The increasing market focus on multi-gigabit line rates and lowering costs (“doing more with less”) not only drives a convergence of communications (i.e., data plane processing) and computation (control plane processing) applications at the platform level, but also presents the technical challenge of understanding optimal hardware-to-software relationships. This implies performance analysis directed towards all platform elements: the processor complex, networking processor, instruction set architecture (ISA), memory and IO interfaces, and interrupt routing. There is a broad range of networking applications that emphasizes the need for these platform elements to complement each other in meeting the overall performance goals.

Performance analysis is the first step in identifying such asymmetries. A closer mapping of the software to the platform hardware is the second step in performance enhancement. Devising more efficient platform elements and hardware interfaces is the third step. We have illustrated this methodology by considering a few real networking applications in their target scenarios. We have demonstrated how to analyze the system-level performance metrics in terms of the underlying hardware (multiprocessing, ISA, IO, and memory bandwidth) and software elements (interrupt processing, SMP synchronization, and ISA). Future work in this area should focus on developing benchmarking standards (e.g., intrusion detection, virus scanning), quantifying control plane processing as a function of data plane processing, and identifying a set of “canonical” platform parameters in a manner that could be directly utilized by networking software developers.

ACKNOWLEDGMENTS

The authors thank several colleagues who contributed extensively to the work presented here: Eswar Eduri, Ravi Gunturi, Frank Hady, Kun Lee, Alex Lopez-Estrada, Pranav Mehta, Uday Naik, Dharmin Parikh, Srikant Shah, Madhukar Tallam, and Raj Yavatkar.

REFERENCES

[1]. <http://www.intel.com/products/benchmarks/desktop/business.htm>

- [2]. Kosiur, David, Building and Managing Virtual Private Networks, John Wiley, New York, 1998.
- [3]. Stallings, William, Cryptography and Network Security: Principles and Practice, 3rd Ed., Prentice-Hall, NJ, 2002.
- [4]. Insolubile, Gianluca, “Paranoid Penguin: An Introduction to FreeS/WAN, Part I,” *The Linux Journal*, September 2002.
- [5]. Bruce Davie and Yakov Rekhter, “MPLS Technology and Applications,” *Morgan Kaufmann*, San Francisco, CA, pp. 49.
- [6]. Werner Bux, Wolfgang E. Denzel, Ton Engbersen, Andreas Herkserdorf, and Ronald P. Luijten, “Technologies and Building Blocks for Fast Packet Forwarding,” *IEEE Communications Magazine*, January 2001, pp. 70-77.
- [7]. Ganesh Balakrishnan and Ravi Gunturi, “MPLS Forwarding Application Level Benchmark Specification Implementation Agreement,” *Revision 1.0, February 2003, NPF benchmark working group*.
- [8]. P. Chandra, F. Hady, R. Yavatkar, T. Bock, M. Cabot, P. Mathew, “Benchmarking Network Processors,” *Proceedings of the 2002 Workshop on Network Processors (NP-1)*.
- [9]. P. Chandra, “IPv4 Forwarding Application-Level Benchmark Implementation Agreement,” *Rev 1.0*.
- [10]. R. Peschi, P. Chandra, M. Castelino, “IP Forwarding Application-Level Benchmark,” *R1.6*, May 12, 2003.
- [11]. E. Rosen, A. Viswanathan, R. Callon, “Multiprotocol Label Switching Architecture,” *RFC3031, IETF*, Jan. 2001.
- [12]. E. Rosen, D. Tappan, G. Fedorkow, Y. Rekhter, D. Farinacci, T. Li, A. Conta, “MPLS Label Stack Encoding,” *RFC3032, IETF*, Jan. 2001.
- [13]. [13]. P. Agarwal, B. Akyol, “Time To Live (TTL) processing in Multi-Protocol Label Switching (MPLS) Network,” *RFC 3443, IETF*, Jan. 2003.

AUTHORS' BIOGRAPHIES

Peter Brink is a senior software engineer at Intel in the Communications Infrastructure Group Technology Office. He received a B.S.E.E. degree from Northwestern University in 1987 and has spent the last sixteen years working on embedded systems. He has been with Intel since 1998 and has worked on USB, multimedia computing, and network performance enhancement. His e-mail is peter.brink at intel.com.

Manohar Castelino is a senior network software engineer at Intel in the Network Processor Division. He received a B.E. degree from KREC India and has worked primarily in the areas of networking and network management. His e-mail is manohar.r.castelino at intel.com.

David Meng is a staff network engineer. He built the first StrongARM Big Endian tool chain based on the open sources in order to compile the Big Endian Linux kernel for the IXP1200 network processor system. He has worked on the SNMP server, Linux kernel, POS/Ethernet RX block, MPLS block, oc-192, and 10G systems. David received a Ph.D. degree in E&CE from New Mexico State University and B.S.E.E and M.S.E.E degrees from Jilin University, China. His email is david.meng at intel.com.

Chetan Rawal is a system architect at Intel, Chandler, Arizona. He works on performance analysis targeted at processor and chipset enhancements for embedded Intel architecture products. In his prior positions at Intel, Chetan worked on various silicon design projects and managed platform validation projects. He received an M.B.A. degree from Arizona State University in 1998, an M.S. degree in Computer Engineering from the University of Massachusetts in 1992, and a B.E. degree in Electronics Engineering from the Birla Engineering College, India in 1987. His e-mail is chetan.rawal at intel.com.

Hari Tadepalli is a staff software engineer at Intel, Chandler, Arizona. He works on performance and architecture enhancements for networking applications. Hari joined Intel in 1996 and contributed to various projects related to IO performance: interconnection fabric of Teraflops supercomputer, PCI on IA32 and IA64 servers, Bluetooth for IA32, and XScale mobile platforms. He received a Ph.D. degree in Computer Science from the University of Delaware in 1996 and an M.S. degree in Computer Science from the Indian Institute of Technology, Chennai in 1987. His e-mail is hari.k.tadepalli at intel.com.

Copyright © Intel Corporation 2003. This publication was downloaded from <http://developer.intel.com/>.

Legal notices at <http://www.intel.com/sites/corporate/tradmarx.htm>.

For further information visit:

developer.intel.com/technology/itj/index.htm