Volume 07 Issue 04 Published, November 14, 2003 ISSN 1535-864X

Intel® Technology Journal

Communications Processing

Twenty years ago, the computer industry was transformed by the emergence of the PC. Today, the telecommunications industry is entering a similar transition. Manufacturers and customers agree that the industry needs to move to modular computing platforms. This issue of Intel Technology Journal (Volume 7, Issue 4) considers various aspects of the silicon, software, and system products and tools that we are building under the umbrella definition of the Intel® Internet Exchange Architecture (Intel® IXA) franchise, and elaborates on some of the concepts and product architectures introduced in the Q4 2002 issue.

Inside you'll find the following papers:

More information, including current and past issues of Intel Technology Journal, can be found at:
http://developer.intel.com/technology/itj/index.htm

Volume 07 Issue 04 Published, November 14, 2003 ISSN 1535-864X

Intel® Technology Journal

Communications Processing

Articles

Preface on Communications Processing, Intel Technology Journal

By Jim Finnegan
Guest Editor, Intel Technology Journal

Defining what one means by "architecture" will inevitably reveal the prejudices of one's functional background and training. For example, a silicon designer might use terms such as *the physical construction* of a computer system from its subsystem components. And if that silicon designer is a student of Hennessy & Patterson's seminal book *"Computer Architecture–A Quantitative Approach"*[1] he/she might consider not just the "instruction set architecture," bu t give equal consideration to performance, physical implementation, power, packaging, and so forth. A software architect might think in terms of program fragments incorporating protocols and interfaces with other programs to form an overall functional application. A networking engineer might define architecture in terms of hardware clients, functional layers, interfaces, and communications protocols used to establish network links and transfer data reliably. Irrespective of one's background, the intrinsic properties of a good architecture may be discerned by asking a series of simple questions:

1. Can a communications system be built cost effectively using this architecture?
2. Do the subsystem components exist?
3. Do the tools exist to build the components and the end system?
4. Is the system robust?
5. Is the system extensible?
6. Is the system performance measurable?
7. How does the system fit with other architectures?

This issue of the *Intel Technology Journal* (ITJ, Vol. 7 Issue 4, 2003) considers various aspects of the silicon, software, and system products and tools that we are building under the umbrella definition of the Intel® Internet Exchange Architecture (Intel® IXA) franchise, and elaborates on some of the concepts and product architectures introduced in the Q4 2002 issue of the ITJ.

In broad terms, one may think of a set of concentric circles, the centre of which is the silicon architecture. Surrounding it are the software architectures and associated software development tools that exploit the hardware capabilities of the silicon. A complete system/platform may be considered the outer circle comprising the top-level hierarchical organization of the subsystem hardware and software components. The nine papers in this issue of the ITJ are so grouped.

System-Level Architecture

The severe downturn experienced over the past few years by Telecom Equipment Manufacturers (TEMs) is well chronicled. Many of those TEMs now realize that their future success is necessarily based upon standards-based *Modular Communications Platforms (MCP)* architectures. The MCP paper recounts the predictable evolution of the MCP architecture, which is now fully defined in a series of specifications called AdvancedTCA* (Advanced Telecom Computing Architecture) for next-generation telecommunications equipment. Well over 100 companies have contributed to these specifications under the aegis of the PCI Industrial Computer Manufacturers Group (PICMG), and inevitably (and arguably correctly) there has been some pragmatic treatment of variables within these specifications. A crucial ingredient of AdvancedTCA is its switched backplane, which is agnostic of fabric technologies (e.g.,

[1] ISBN 1-55880-069-8

® Intel® Internet Exchange Architecture (Intel® IXA) is a trademark of Intel Corporation or its subsidiaries in the United States and other countries.

InfiniBand*, PCI Express*, Advanced Switching, etc.). The paper "*Fabrics and Application Characteristics for AdvancedTCA Architectures"* examines some of the attributes of these technologies.

While the capital costs of MCPs are an important consideration, the service provider is much more concerned about the Total Cost of Ownership (TCO), which translates directly into high availability (99.999%), serviceability, manageability, configurability, and scalability. Considerable effort has been made to provide operating system software components that meet these criteria. The paper *"AdvancedTCA/CGL Advantage: HA and TCO"* describes some of the properties of Intel's Carrier-Grade Linux and High Availability (HA) middleware such as fault detection/fault management, fast policy-based recovery, which when operating in tandem with ATCA hardware features such as the Shelf Manager, meet the service provider's TCO goals.

Software Architectures
The ubiquity of TCP/IP as the data communications protocol of choice is beyond dispute. The advent of 10 Gbps Ethernet technology presents significant challenges to those applications that had previously been able to process TCP/IP packets at lower speed. The paper *"Addressing TCP/IP Processing Challenges using the IA and IXA Processors"* elucidates some of the challenges (e.g., memory latency, connection look-up requirements, checksum calculation) and illustrates how the highly parallel IXP network processor is able to support full line rate 10 Gbps TCP/IP packet processing requirements. While 10 Gbps TCP/IP packet processing may be considered "today's problem," it is somewhat obvious that traffic trends in the network infrastructure will conspire to present a new set of problems in the not-too-distant future. For example, the volume of control traffic is likely to grow significantly due to (a) exponential growth in the number of network elements, and (b) increasing requirements for millisecond convergence times upon link or node failures. In the paper, *"Distributed Control Plane Architecture for Network Elements*," the authors have conceived a clever distributed control plane architecture, which allows a control plane protocol to be split between the conventional control processor and other available processing elements (such as unused bandwidth on the data plane processor).

Intel remains fully committed to standardizing the IXA software framework as it evolves. The philosophy behind the software framework is to promote wholesale development of modular, reusable software building blocks with the inherent economic and quality benefits of such reuse. A recent extension to the framework includes support for multicast and packet fragmentation, as described in the *"Advanced Software Framework, Tools and Languages for the IXP Family"* paper. In that paper, the authors describe some of the new tools, including the next-generation compiler and the packet generator/profiler, which enable packet-centric debugging. The new compiler assumes a data flow model, with functional fragments that are reasonably independent.

Hardware Architectures
Ultimately, it is necessary to empirically measure the performance of a platform or system. The paper *"Network Processing Performance Metrics for IA- and IXP-Based Systems"* describes a benchmarking methodology that evaluates the performance impact of architectural decisions at multiple levels. It should be fairly obvious by now that there is a need to provide dedicated packet processors adjunct to the more traditional application processors. This, after all, represents the genesis of the IXP network processors. A more recent and subtle observation is that throughput and latency considerations demand increased state-sharing between the application processor (IA) and the packet processor (IXP) as line rates increase. The implications are discussed in the paper *"Enterprise Edge Convergence: Packet Processing and Computing Silicon in the Data Center."*

Finally, some of the IXA components are described in some detail. The IXFxxx framers/mappers/cross-connects, when used in conjunction with the IXPxxx family of network processors, enable the design of richly featured scalable multi-function multi-bandwidth line and service cards for edge platforms. The paper *"Scalable Intel® IXA and its Building Blocks for Networking Platforms"* provides an overview of some of these components and their interfaces as well as a decomposition of the key ingredient of the Network processors, the microengine, which is a 32-bit RISC processor supporting an array of specialized packet-processing features.

You can read past issues at http://developer.intel.com/technology/itj/archive_new.htm.

You can also subscribe thru a simple registration form at http://www96.intel.com/cme/showSurv.asp?formID=1019&actv=REG.

Foreword
Communications Processing

By Sean Maloney
Executive Vice President, Intel Corporation
General Manager, Intel Communications Group

Twenty years ago, the computer industry was transformed by the emergence of the PC. Over a period of 5-10 years, the "Old Computer Industry" – horizontally organized with top-to-bottom proprietary suppliers like IBM and DEC – was supplanted by the "New Computer Industry" – built around mass-produced standards.

The telecommunications industry, buffeted by three years of severe recession, is now entering a similar transition. However, history will not fully repeat itself. Software supply is still diverse, and technical innovation is still proceeding at a pace sufficient for significant product differentiation. But there is a strong consensus among manufacturers and their customers that we need to move to modular computing platforms.

An early sign of this is the groundswell of support for the Advanced Telecom Computing Architecture (AdvancedTCA*) initiative, which defines an industry-standard modular computing platform. AdvancedTCA was designed to meet the requirements of the communications sector. These requirements include attributes such as "simplicity," "capacity," "performance," "reliability," "serviceability," "compliancy," and "security" – all of which are critical to the industry. Intel is well positioned to tackle this market with a full suite of ingredients from its family of IXPxxx network processors, IXFxxx framers and mappers, and carrier-grade rack-mount servers using the Intel® Xeon™ and Intel Itanium® 2 processors. We also recently launched the Intel NetStructure AdvancedTCA platform and Carrier-Grade Linux* operating system.

This issue of Intel Technology Journal examines these technologies in depth through the eyes of Intel's communications architects.

Why should this matter to equipment manufacturers? Chuck Byers of Lucent Technology and a Bell Labs Fellow stated it most eloquently in his recent paper, "From Standards to Platforms to Products to Profit," when he said, *"Platforms are the most valuable if they are based on open industry standards, supported by a vital multiple source ecosystem (like AdvancedTCA."*

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

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

Modular Communications Platform

Matthew M. Boam, Intel Communications Group, Intel Corporation
Jay Gilbert, Intel Communications Group, Intel Corporation
Tisson K. Mathew, Intel Communications Group, Intel Corporation
Karel Rasovsky, Intel Communications Group, Intel Corporation
Raj Sistla, Intel Communications Group, Intel Corporation

Index words: AdvancedTCA, bladed shelf, Modular Communications Platform (MCP), Rack-Mounted Servers (RMS), Telecommunications Equipment Manufacturers (TEMs), Home Location Register (HLR)

ABSTRACT

After years of severe downturn in business activity, the communications industry is now embracing standards-based Modular Communications Platforms (MCPs) as a way to improve profitability while the market recovers. Eclipsing traditionally closed, proprietary communications systems, the adoption of MCPs promises Telecom Equipment Manufacturers (TEMs) and Service Providers (SPs) smaller initial outlays and reduced Total Cost of Ownership (TCO), while at the same time making it easier to develop and run applications. The MCP paradigm is rooted in several industry-wide standards initiatives, and spans a broad ecosystem of building blocks including silicon, boards, chassis, operating systems, middleware, applications, and more.

Historically, the shift toward hardware modularity began with Rack-Mounted Servers (RMSs) that were optimized for compute-centric applications in enterprise networks and back-office systems. To accommodate the extreme requirements of central office environments, however, a more flexible, reliable, modular, scalable, and higher performance approach was needed. Thus, the concept of a "bladed shelf" was born. Recently, the industry represented in the PCI Industrial Computer Manufacturers Group (PICMG) standardized a bladed form factor in a series of specifications known as AdvancedTCA[*].

AdvancedTCA takes a holistic approach to High Availability (HA) and five 9s (99.999 %) reliability required in carrier-grade equipment through the integration of features for resiliency, redundancy, serviceability, and manageability. AdvancedTCA also lays a solid foundation for scalability of system performance and capacity, so that application processors, network processors, Digital Signal Processor (DSP) boards, and storage boards can be hot-swapped on demand with guaranteed interoperability, maintainability, and reliability. With its high emphasis on low-cost manufacturing, AdvancedTCA-based platforms will lower the cost of network elements.

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

INTRODUCTION

Driven by better economics, Telecom Equipment Manufacturers (TEMs) increasingly embrace a standards-based modular approach to building communications systems. These Modular Communications Platforms (MCPs) are founded on several industry standards. They are architected to meet the highest reliability, stability, and scalability requirements found in applications from the edge to the core of the communications network.

MCPs are not form-factor specific. Rack-Mounted Servers (RMS), CompactPCI, and AdvancedTCA are all examples of standard MCPs. Of these, AdvancedTCA is best suited to meet the demands of carrier-grade infrastructure for the next decade. Founded on the requirements of the communications infrastructure, PCI Industrial Computer Manufacturers Group 3.0 (PICMG 3.0) specifies the electro-mechanical, interconnect, and management aspects for building a modular, reconfigurable shelf. The large form factor of AdvancedTCA blades provides ample architectural headroom for next-generation silicon and applications. Switched backplanes, agnostic of fabric technologies (e.g., InfiniBand*, PCI Express*, StarFabric, facilitate extensive bandwidth scalability for a broad range of applications. The robust AdvancedTCA shelf management is based on the Intelligent Platform Management Interface (IPMI) 1.5 specification, and it enables hot swapping, inventory information management, power distribution, and management facilities.

In this paper, we explain how to design MCPs on standard modular building blocks—silicon, hardware, operating systems, and middleware—using the 3G wireless application (a Home Location Register (HLR)) as an example. Standardization and broad adoption of MCPs is linked to reliable interoperability. With this objective, PICMG periodically sponsors interoperability events, focused on validating early vendor implementations.

OVERVIEW OF THE MODULAR COMMUNICATIONS PLATFORM CONCEPT

Transition to Modular, Standards-Based Communications Infrastructure

Traditionally, Network Equipment Providers (NEPs) developed key building blocks of communications systems in-house, to satisfy system requirements for high reliability, stability, and performance. The lower layers of these purpose-built platforms consisted of a proprietary operating system (OS), ASICs, and other purpose-built silicon, custom boards, specialized chassis, and purpose-built APIs and application software (SW).

Proprietary platforms designed around vertical applications are very expensive to build and maintain. As the industry matures, this is further compounded by increased competition, calling for greater differentiation, faster time-to-market (TTM), the need for outsourcing, and the increased pace of technological innovation. Consequently, the communications network is now undergoing a natural transition to a modular, standards-based model, based on Modular Communications Platforms (MCPs). MCPs are standards-based communications infrastructure platforms and building blocks designed to deliver reduced expenditure, solution flexibility, TTM, and vendor choice (see Figure 1).

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

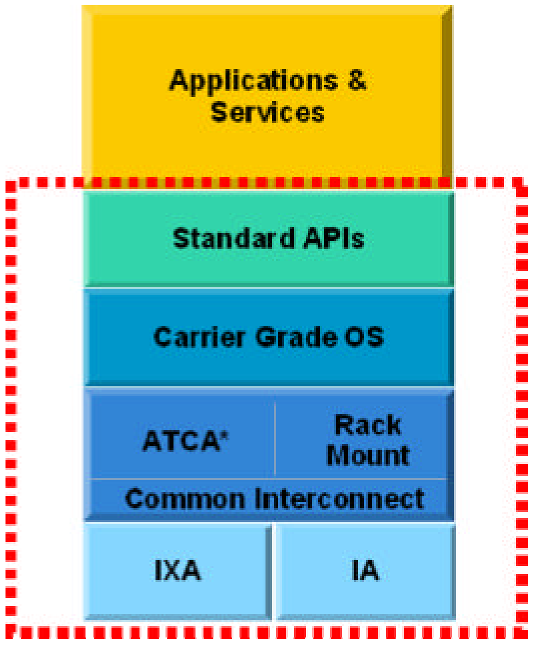


Figure 1: Elements of MCP

The result of this transition will be more investment at the application and service layers of the network infrastructure, enabling more innovative, lower-cost, revenue-generating services faster for service providers and lower cost, time-to-money platforms for NEPs.

Modular Communications Platform Delivers Flexibility, Modularity, and Scalability

Supported by a growing ecosystem of standards-based suppliers, MEPs offer multiple benefits to both NEPs and network operators.

As NEPs will increasingly outsource lower-level MCP building blocks to a rich ecosystem of suppliers, they will be able to focus on their core competencies in developing application software and will be able to improve their responsiveness to changes in market demands and to economic fluctuations.

Standards drive predictability and broad platform investment, enabling innovative, lower-cost solutions and greater vendor interoperability. A broad selection of interoperable building blocks such as IO blades, compute blades, switch hubs, and chassis will provide a faster integration of communications services or solutions, thus shortening the lengthy custom product cycle.

One of the most critical benefits for Telecom Equipment Manufacturers (TEMs) and operators alike is the re-use of standard modular components across multiple applications. For example, TEMs could streamline all Radio Access Network elements such as a Radio Network Controller (RNC), a Serving GPRS Support Node

(SGSN), Gateway GPRS Support Node (GGSN), etc. on a single common platform, with the same standards-based chassis, line cards, and switch fabric blades. This will have a positive impact throughout the communications value chain. TEMs will be able to exploit the cost benefits of volume manufacturing and purchasing, the cost savings of reusing components, and the reduction in R&D expenses for platform development and maintenance. For operators, this will result in reduced deployment and maintenance costs, through lower inventories and easier network deployment and maintenance. Standard common interfaces across network components—hardware and software—enhance automation and management of disparate systems through a single management model.

Elements of MCPs

MCPs are built on technologies encompassing silicon, hardware, software, and middleware layers of the stack.

Since MCPs will be found in network elements ranging from enterprise edge to carrier-grade systems at the core of the network, the processing requirements have to address user plane, data plane, and control plane demands. User plane applications, such as those in server-based infrastructures, require compute-intensive processing power. In data plane processing, a new generation of software programmable network processors is gaining a foothold. These generic network processors deliver the "fast path" capabilities required to process packets at wire-speed per service policies. Enabling network services also requires control tasks such as setting and adjusting service policies, something that is increasingly implemented on control processors for flexible, cost-effective design.

MCPs are not form-factor specific. They encompass hardware options to suit the necessary requirements and timing of a particular build-out. Hardware form factors include AdvancedTCA and CompactPCI chassis and blades and Network Equipment-Building System (NEBS)-compliant Rack-Mounted Servers (RMSs). Software modular components include a carrier-grade operating system, high-availability (HA) middleware APIs (through the Service Availability Forum) and high-performance, scalable IO interconnects (advanced switching based on PCI Express architecture).

To meet the rigorous requirements of telecommunications, OSs must be robust, reliable, and provide the advanced management and diagnostics common in today's carrier-grade network equipment. MCPs offer choice of carrier-grade OSs, according to the targeted application.

The primary goal of standard, HA middleware APIs is to facilitate an open programming environment resulting in simplified upgrades and greater middleware vendor interoperability. The Service Availability Forum offers standard API specifications to the telecommunications industry so that the best middleware solutions are implemented in a flexible programming environment.

To meet the performance demands of current telecommunications applications, MCPs require high-performance, scalable IO technology, and interconnects based on industry-accepted standards. Advanced switching, based on industry-standard PCI Express architecture for example, allows for a feature-rich, scalable architecture targeted at both volume switch fabric and chip-to-chip interconnects within the traditional communications and embedded market segments.

Modular Platform Choices

MCPs have evolved over the past six years into a variety of flavors. The basic impetus behind the development of MCP solutions was to ensure HA in carrier-grade systems and to maximize the sharing of resources, such as power and cooling.

Rack-Mounted Servers

This early flavor of modular platforms grew as a result of demand in the enterprise (data center) and Telco environments for stackable hardware for running entry-level servers for storage, hosting, and other purposes. The biggest advantage of RMSs was their small size (typically 1U or 2U in size) and hence their flexibility of deployment. In a strict sense, RMSs do not share power or cooling. RMS solutions are commoditized, and for more sophisticated and HA-demanding applications, they have paved the way for technologies like CompactPCI and AdvancedTCA.

CompactPCI

CompactPCI started as an adaptation of the Peripheral Component Interconnect (PCI) Specification 2.1 or later for industrial and/or embedded applications (e.g., Telecom) requiring a more robust mechanical form factor than desktop PCIs. CompactPCI used industry-standard mechanical components and high-performance connector technologies to provide an optimized system intended for rugged applications. CompactPCI provided a system that is electrically compatible with the PCI specification, allowing low-cost PCI components to be utilized in a mechanical form factor suited for rugged environments. CompactPCI technology was proposed by the PCI Industrial Computer Manufacturers Group (PICMG)

Specification 2.0 and then the various 2.x specifications were drafted to augment the base specification.

CompactPCI caters to customer segments that require these capabilities [13]:

- 33 MHz PCI performance
- 32- and 64-bit data transfers
- 8 PCI slots per bus segment
- 3U and 6U form factors for boards

Since CompactPCI borrowed heavily from PCI, it retains the notion of a system master and peripherals. A CompactPCI system is composed of one or more CompactPCI bus segments. Each segment is composed of up to eight CompactPCI slots with 20.32 mm (0.8 inch) board center-to-center spacing. Each CompactPCI bus segment consists of one system slot, and up to seven peripheral slots. The system slot provides arbitration, clock distribution, and reset functions for all boards on the segment. It is responsible for performing system initialization by managing each local board's IDSEL signal. The peripheral slots may contain simple boards, intelligent slaves, or PCI bus masters.

CompactPCI was the first revision of the PICMG specifications that comprehended modularity and MCP. In CompactPCI, modularity is achieved by utilizing various Eurocard form factors. CompactPCI also comprehends the notion of hot swapping the modules in the system. This was originally implemented by "pin staging" i.e., pins of multiple lengths that make and break contact with the backplane jumpers at different times.

CompactPCI implements a "keying" mechanism to differentiate 5 V or 3.3 V signaling operations. The keying mechanism is designed to prevent a board built with one technology (with one set of voltage specs) from being inserted into a system designed for another technology (with different specs). The keys are mechanical keys that are inserted on the backplane pin-out and they are color coded to denote the appropriate technology that the slot is meant to support.

Today CompactPCI is a well-adopted, mature technology although it has its own shortcomings. For instance, the system management support is very rudimentary in CompactPCI, and the specification does not mandate a standard set of commands, interaction between entities, etc.

Advanced Telecom Computing Architecture

In January 2003, PICMG announced the availability of a new series of specifications, called AdvancedTCA, for next-generation telecommunications equipment, with a new form factor and based on switched fabric architectures. AdvancedTCA was the largest specification effort in PICMG's history, with more than 100 companies participating in its development. This series of specifications incorporates the latest trends in high-speed interconnect technologies, next-generation processors and improved reliability, manageability, and serviceability. Companies participating in the AdvancedTCA effort have brought a wide range of knowledge of the industry. They include Telecommunications Equipment Manufacturers, (TEMs) board and system-level vendors, computer Original Equipment Manufacturers (OEMs), software companies, and chassis, connector, and power supply vendors.

AdvancedTCA aims to start where CompactPCI left off and to take MCP to the next level. It also standardizes the concepts that were left unaddressed in CompactPCI, which led to some ambiguity. In AdvancedTCA, the concept of system management was given a great deal of importance. AdvancedTCA mandates the notion of a shelf manager as well as that of an Intelligent Platform Management Interface (IPMI)-capable controller on each entity on the management bus. The increased focus on management renders the AdvancedTCA-based MCP solutions capable of high MTBF (Mean Time Between Failure).

In the next section we delve into the advantages of AdvancedTCA in greater detail.

ADVANCEDTCA MAIN FEATURES AND VIRTUES

Mechanical Overview

This section showcases the essential mechanical features of AdvancedTCA-compliant platforms. The basic elements include front boards (compute boards, line cards, switches, and other components) and Rear Transition Modules (RTMs) (see Figure 2) which provide user-defined input and output (IO) connectivity to the companion front board from the rear of the AdvancedTCA chassis. The backplane shown in Figure 5 shows connector interfaces for power distribution and IO connectivity between chassis boards. Additionally, Chassis Management IO via the Intelligent Platform Management Bus (IPMB), the Update Channel interface and Synchronization Clock busses are also depicted. All these electrical and mechanical interfaces are defined in the PICMG 3.0 specification and are discussed further below.

The front board is 8U high, 30.48 mm wide, and approximately 280 mm deep. The front board has to include a face plate and two handles; which are mandatory features. On the front board, three connector zones are defined, namely Zone 1 for power connection and shelf management, Zone 2 for data transport interface, and Zone 3 for user-defined IO interconnect.

The front boards within a shelf are defined as being in the standard orientation when they are positioned vertically. The RTM consists of a 322.25 mm x 70 mm PCB, 8U x 6 HP face plate. Backplanes are from 3 to 8 mm thick.

The number of front-board slots accommodated by the backplane ranges from 2 to 16. The front-board form factor has been designed for use in a 12U vertical shelf with a 600 mm frame depth. Cable management is determined at the shelf and front-board/RTM level. Within each of these three areas, there are three types of cable management. They are power cable management, IO copper cable management, and IO optical fiber cable management. Front boards get filtered air; shelves provide filtering of air using an air filter tray. Physical slot numbering is from left to right for front boards as viewed from the front in the standard orientation, and from bottom to top in the horizontal orientation. In a 19" shelf, a subrack with front boards oriented in the vertical direction can accommodate up to 14 front boards. In a 600 mm-wide European Telecommunications Standards Institute (ETSI) shelf or a 23" shelf, a subrack with front boards oriented in the vertical direction can accommodate up to 16 front boards.

Backplane

Figure 2 shows a 14U AdvancedTCA backplane.

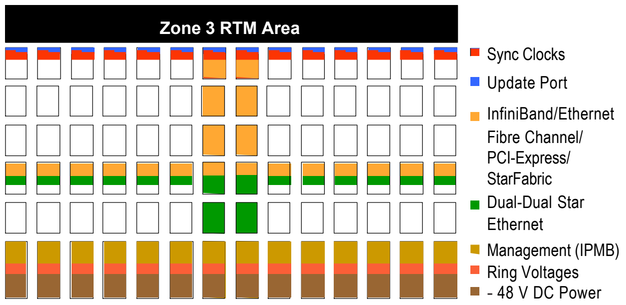


Figure 2: 14-node AdvancedTCA backplane [8]

Shelf (Chassis)

Figure 3 shows a 14U AdvancedTCA shelf fully populated with blades.

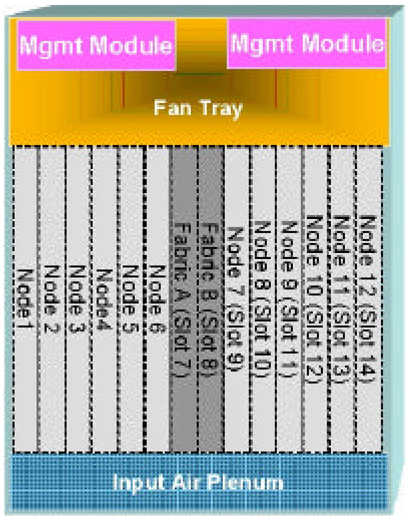


Figure 3: AdvancedTCA shelf [8]

Front Board and RTM

Figure 4 presents a front board and an RTM.

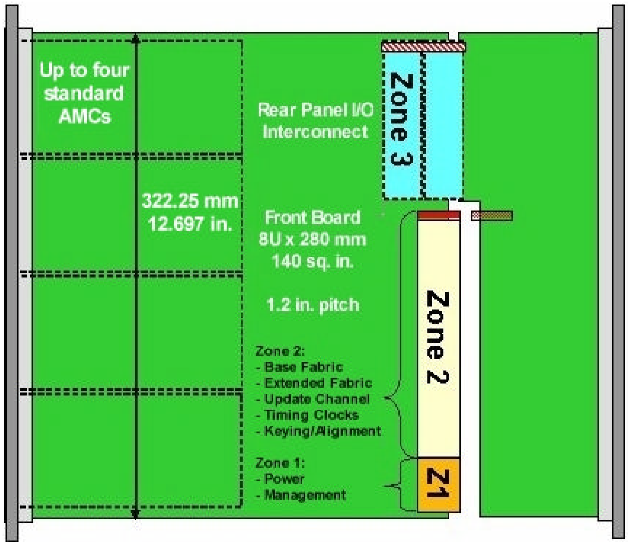

Up to four standard AMCs
Rear Panel I/O Interconnect
Zone 3
322.25 mm 12.697 in.
Front Board 8U x 280 mm 140 sq. in.
1.2 in. pitch
Zone 2: - Base Fabric - Extended Fabric - Update Channel - Timing Clocks - Keying/Alignment
Zone 2
Zone 1: - Power - Management
Z1

Figure 4: AdvancedTCA front board w/ RTM [8]

Interconnects

The AdvancedTCA interconnect framework is comprised of (1) the physical connector used to mate boards and backplanes, (2) the mapping of signals to that connector, and (3) the routing of those signals among boards across the backplane. The AdvancedTCA front board supports five different communication interconnects that can be used for control, data, and management traffic. These interconnects are illustrated in the subsections 4.1

through 4.5. The data transport connector zone (Zone 2) is comprised of up to five "ZD" connectors per board/slot. The ZD connector is a highly reliable connector defined in the PICMG 3.0 specification for use in Zone 2 of a standard AdvancedTCA board. It supports 40 signal pairs on each connector and provides 200 differential signal pairs for an AdvancedTCA board to establish connectivity with up to 15 other boards via a common backplane.

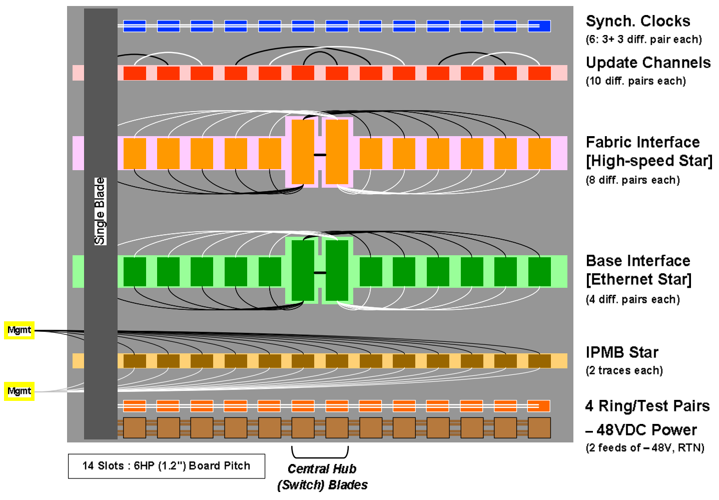


Figure 5: AdvancedTCA interconnects [8]

Base Interface

The Base interface is comprised of 64 differential signal pairs and it contains up to 16 Base channels for a total of 64 possible signal pairs per board/slot. A Base channel (4 signal pairs) can be used to support a 10/100/1000BASE-T port. The Base interface is always configured as a dual star with redundant switch. When a board is connected to both switches the Ethernet configuration will be a dual-dual star. If a designer wishes to connect boards using InfiniBand on the Base interface, a one differential signal pair (1X, 2 differential signal pairs) configuration of InfiniBand interconnect can be used. This achieves a signal rate of 2.5 Gbps in each direction. If InfiniBand is supported, the designer must be sure that Base interface E-keying is supported by the shelf manager and boards.

Fabric Interface

The Fabric Interface consists of 120 differential signal pairs. The fabric interface is comprised of 15 channels providing connectivity among up to 16 boards in a full mesh or star configuration. The fabric interface can be used in a variety of ways by boards and backplanes to meet the needs of many applications. The fabric interface can be used to run various options for IO fabrics including Ethernet (*PICMG 3.1*), InfiniBand (*PICMG 3.2*), StarFabric (*PICMG 3.3*), or PCI Express (*PICMG 3.4*). The fabric interface can support only one protocol at one time. E-Keying is used to guarantee this exclusivity.

Utilizing 4X 10 GBps (Gigabits/sec) InfiniBand connections in the configuration with 14 boards, the backplane supports an aggregate bandwidth of 1.8 TBps (Terabits/sec) full duplex (3.6 TBps total bidirectional bandwidth). A backplane supporting 16 boards would provide about 2.1 TBps. While multiple fabrics are envisioned by the specification, Ethernet has significant presence in the market place today and it is likely that Advanced Switching technology will emerge as the dominant general-purpose IO fabric in the longer term, while other technologies will address special-purpose applications.

Update Channel Interface

The Update Channel Interface is comprised of 10 differential signal pairs in a point-to-point connection between two boards/slots. Update channel can be used by two boards to form a redundant pair and can communicate status information (enabled vs. fail and active vs. standby) in a simple and robust manner. Using the update channel, two boards can exchange a large amount of traffic with requirements that cannot be met by the switching fabric. The most common example is a pair of interface boards on a SONET/SDH ring. Synchronous traffic switching is a very specialized application that packet-based fabrics do not support. The delay requirements (fewer than 25 microseconds interface to interface) are not compatible with packetization and queuing schemes. The solution is to allow the two boards to transport their synchronous pass-through traffic over the update pairs and bypass the fabric.

Synchronization Clock Interface

The synchronization clock interface provides a set of clock buses to enable applications that require the exchange of synchronous timing information among multiple boards in a shelf. Unlike the CompactPCI architecture, the AdvancedTCA architecture does not define a time division multiplexed bus, such as the H.110 bus, that would require synchronous timing signals internally for bus access. Yet, many telecommunications applications using the AdvancedTCA architecture still need to interface to external networks that require strict timing relationships between multiple interfaces and the external network, such as Plesiochronous Digital Hierarchy **(PDH)** networks and SONET/SDH (Synchronous Digital Hierarchy). These are communications networks in which different parts of the network are almost, but not quite, perfectly synchronized.

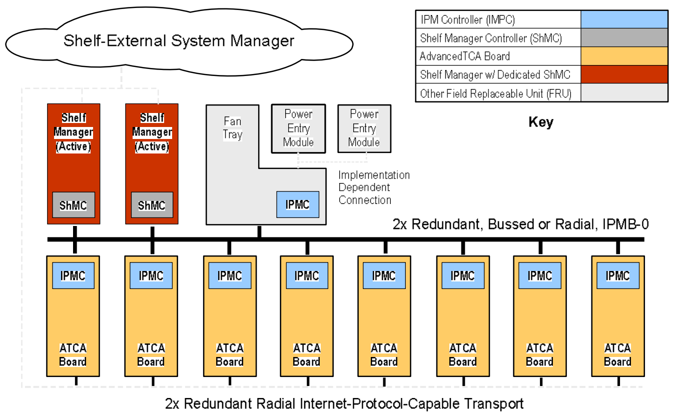
Management Interface

The management interface in Zone 1 is a two-way redundant implementation of the Intelligent Platform Management Bus (IPMB) based on a two-wire serial bus. In AdvancedTCA shelves, the main IPMB is called IPMB-0 and is implemented on either a bussed or radial basis. Each entity is attached to IPMB-0 via an IPM controller, the distributed management controller of the IPMI architecture to the shelf manager (a special IPM controller).

Management Interface

A logical diagram of the AdvancedTCA management architecture is shown in Figure 6.

Figure 6: AdvancedTCA management architecture


Shelf-External System Manager
IPM Controller (IMPC)
Shelf Manager Controller (ShMC)
AdvancedTCA Board
Shelf Manager w/ Dedicated ShMC
Other Field Replaceable Unit (FRU)
Key
Shelf Manager (Active)
Shelf Manager (Active)
ShMC
ShMC
Fan Tray
Power Entry Module
Power Entry Module
Implementation Dependent Connection
IPMC
2x Redundant, Bussed or Radial, IPMB-0
IPMC
ATCA Board
2x Redundant Radial Internet-Protocol-Capable Transport

[1]

The AdvancedTCA system management protocol is IPMI. AdvancedTCA generalizes the concept of the platform manager into an entity called the shelf manager Baseboard Management Controller (BMC), in the IPMI world. The AdvancedTCA management responsibilities are distributed among IPMCs that are responsible for the management board and its subcomponents, and redundant shelf managers. The AdvancedTCA specification describes two extremes on the bus topologies that are allowed within the specification. The first is a dual bus topology where all entities interact over a pair of shared busses. The other extreme is that all IPMCs connect in a dual star (with twin busses on each star leg) topology to the shelf managers. AdvancedTCA allows systems with either 1) all components operated by a single owner, or 2) components operated by multiple owners, i.e., a data center model. The system manager can be considered as a higher-level, high-speed service for boards that need TCP/IP-based management services such as remote booting, Simple Network Management Protocol (SNMP) management, remote disk services, and other IP-based services.

The purpose of the shelf management system is to monitor, control, and ensure proper operation of AdvancedTCA boards and other shelf components. The shelf management system watches over the basic health of the system, reports anomalies, and takes corrective action when needed. It can retrieve inventory information and sensor readings as well as receive event reports and failure notifications from boards and other intelligent Field Replaceable Units (FRUs). A shelf manager manages the usage of shelf power and cooling resources to ensure that the shelf remains within its design limits. Boards must also report their backplane interconnect types to system management before the interconnects are enabled. The shelf manager uses the information to detect mismatches and will only instruct boards to enable links that are compatible so that boards do not damage each other. This feature is also known as E-Keying. The shelf manager is also responsible for security and Quality of Service (QoS) in a multi-tenant environment.

Building a Network Element on MCP

Network elements for MCP refer to many of the different applications that the MCP architecture can support and of course, the building blocks that make up the total solution for each MCP application. Consider, for example, a Home Location Register (HLR) network element. An HLR forms a critical component of a wireless core telecommunications server in that it supports the SS7 database used in mobile networks that identify and verify a subscriber. It also contains subscriber data related to the features and services supported by the network. The HLR is located in the Service Control Point (SCP) of the cellular provider, which also holds databases in which customer-specific information used by the network to route calls is stored (such as 800 number routing, subscriber services, calling card validation). A typical 2.5 G wireless network including HLR is shown in Figure 7.

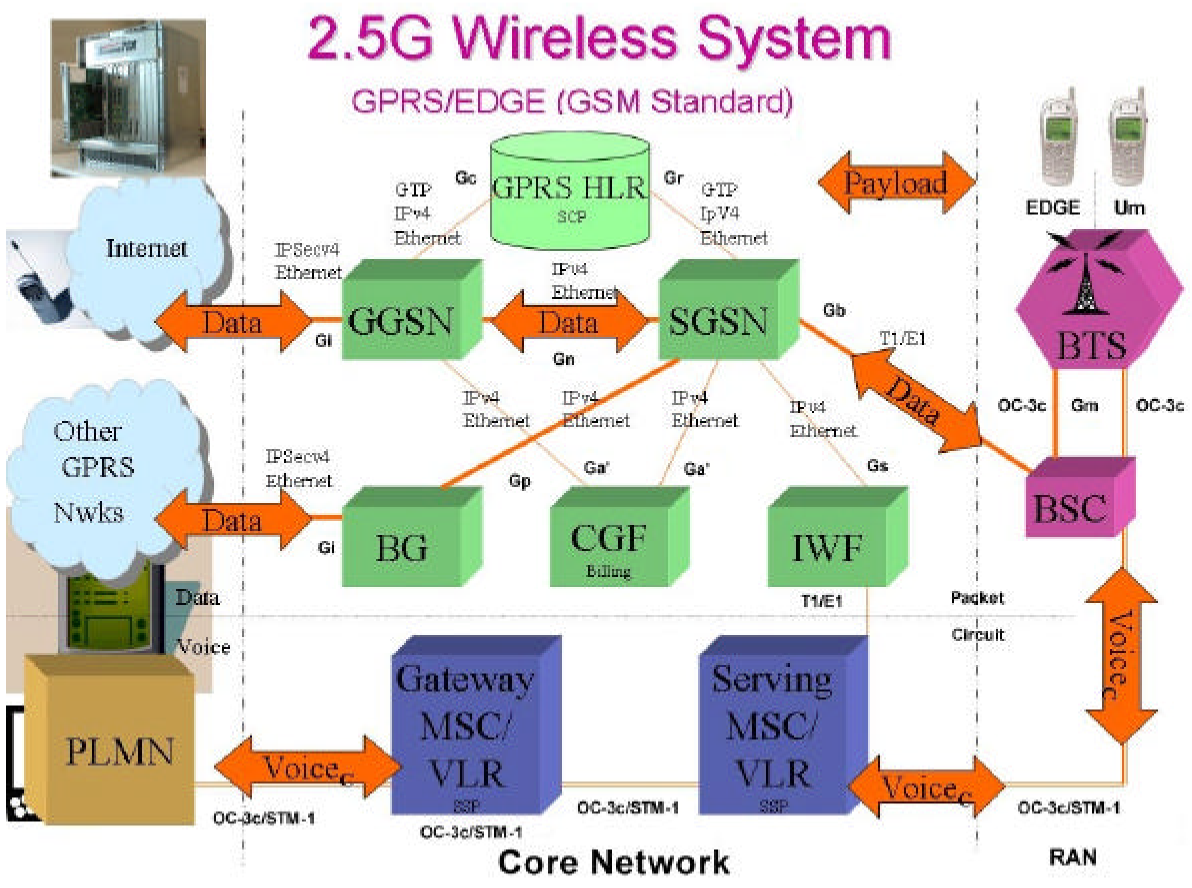


Figure 7: Typical 2.5 G wireless network with HLR

To go deeper into this example, you can define the number of HLR subscribers in terms of low, medium, and high densities and then map them against your HLR equipment needs. A low-density HLR assumes approximately 1 million subscribers, a medium-density HLR assumes about 3 million, and a high-density HLR assumes about 8 million subscribers. In our example, we focus on mid density of 3 million and assume the carrier is looking to the new generation of telecom equipment based on the MCP/AdvancedTCA standard. Memory storage requirements for 3 million users at an average memory usage of 2 KB per user equals 6 GB of RAM with a throughput of > 54 MB/sec. This database would reside on its own Single Board Computer (SBC) while the HLR application, on a separate SBC to support 3 million users, would require approximately 8 GB of total RAM. In order to support the strict HA requirements (often referred to as the five 9s or 99.999%) of carriers and operators, TEMs have been migrating to the Linux operating system. These hardened, highly reliable OS platforms are referred to as CGL or Carrier-Grade Linux with offerings from many ecosystem vendors.

For storage, an external Fibre Channel array can be attached to an SBC AdvancedTCA-based blade, which is expected to be in production and available in Q4, ’03. The Zone 3 connector (top rear) on an AdvancedTCA standards-based board can be used to support this feature unless the system integrator wishes to route the cabling out the front, in which case some of the new SBCs have front Fibre Channel connectivity. AdvancedTCA is unique in its support of either front or rear connectors and expansion capabilities. As an alternative, an in-chassis storage solution such as Small Computer Systems Interface (SCSI) disk-based blades may be used. Either way, there are excellent, cost-effective solutions available on the market to support this application.

HLR network element configurations also require signaling support to interface with the larger network. For signaling, many of the new SBC boards support Processor Mezzanine Cards (PMCs) which enable expansion and interconnect capabilities. For our 3 million-user example above we can expect to need 12 + 12 ports of E1/T1 b/w.

The ecosystem currently focused on AdvancedTCA/MCP-based products is quickly developing offerings in support of many network elements including HLRs. The industry collaborates on meeting strict interoperability requirements through a series of on-going workshops and industry-wide testing events. Current progress shows the developing ecosystem is well on track to meet the challenging requirements of scalable and cost-effective solutions to growing telecommunications infrastructure requirements.

DEFINITIONS

Core Networks (CN): back office network elements responsible for providing functions responsible for connecting the mobile subscriber to the public land mobile network (PLMN) including call control, delivery, billing, etc. (Trunking is used for connectivity.)

Public land mobile network (PLMN): a telecommunications network for mobile units, referred to as mobile stations or mobile phones.

Radio Access Networks (RAN): "in the field" network elements responsible for delivering the call via the "air interface" (a virtual line) that is composed of cell towers, cell phones, etc.

CONCLUSION

As discussed throughout the paper, the Modular Communications Platform (MCP) adds significant value to the ecosystem through reductions in capital and operational expenditures for operators while helping to grow a rich ecosystem of Telecom Equipment Manufacturers (TEMs) focused on MCP building blocks. This standards-based, building-block approach to the needs of carriers and operators helps move the ecosystem away from proprietary solutions to more scalable, open architected platforms. The net result for operators as they migrate to MCP is to allow them to focus on higher-level value-added differential services while increasing the reliability of their networks through industry-accepted standards and telecommunications-grade building-block products. As the industry matures and competition increases, operators will increasingly look for opportunities to enhance their networks and lower their capital and operational costs. MCP affords them such an opportunity focusing the industry on a broadly adopted industry standard that embraces open architectures and the support of many ecosystem players. As a key ecosystem participant, Intel is supporting AdvancedTCA from both the standards development perspective and through a broad array of silicon, board, chassis and software product offerings. Intel is committed to open architectures and standards and strongly endorses the efforts of PICMG with AdvancedTCA and MCPs.

REFERENCES

[1] PICMG 3.0 "AdvancedTCA Specification 1.0," December 2002.

[2] "InfiniBand Architecture, Volume 1, General Specifications," *InfiniBand Trade Association*, 2000.

[3] "InfiniBand Architecture, Volume 2b. Mechanical, Cable and Optical Specifications," *InfiniBand Trade Association*, 2000.

[4] PICMG 3.1 "Specification for Ethernet Fabric Interconnects," December 2002.

[5] PICMG 3.2 "Specification for InfiniBand Fabric interconnects," January 2003.

[6] PICMG 3.3 "Specification for StarFabric interconnects" January 2003.

[7] PICMG 3.4 "Specification for PCI Express Interconnects," January 2003.

[8] Kevin Bross, Intel Corporation, "New Generation Communications Systems on AdvancedTCA," Intel Developer Forum, Fall 2002.

[9] Service Availability™ Forum "Platform Interface Specification," January 2003.

[10] Daniel M. Pressel, "Scalability vs. Performance," *U.S. Army Research Laboratory*, Aberdeen Proving Ground, MD.

[11]"Criteria for Evaluating High-Speed Serial Switched Interconnects," *COTS Magazine,* March 2003 Edition.

[12] Bustronic Corporation: "The Power of the Interconnect Evolution."

[13] PICMG 2.0 "Compact PCI Specification 3.0," October 1999.

AUTHORS' BIOGRAPHIES

Matt Boam is a software engineeer in the Embedded IA Division (ICG/NPG) at Intel. His recent work has focused on server mangement solutions. He received his B.S. degree from the University of Idaho. His e-mail is mathew.m.boam at intel.com.

Jay Gilbert is a senior technical marketing manager within Intel's Communications Infrastructure Group. He is responsible for standards development, customer training and technology evangelism in support of Intel's AdvancedTCA communications infrastructure development efforts. Mr. Gilbert has been with Intel Corporation for more then 12 years and holds a B.S.E.E. degree from the Oregon Institute of Technology and an

M.B.A. degree from Portland State University. His e-mail is jay.gilbert at intel.com.

Tisson Mathew is a research scientist with Intel System Software Lab where he works on Platform Virtualization and Utility Computing. His other research interests are advanced computer architecture, distributed systems, networking and operating systems. He has been with Intel since 1999. His e-mail is tisson.k.mathew at intel.com.

Karel Rasovsky is a technical marketing engineer in Intel's Marketing and Platform Programs. He is currently responsible for introducing Modular Communications Platforms to the market. Prior to Intel, he was a product marketing manager at Lucent Technologies, focusing on converged voice and data solutions for next-generation networks. His other roles at Lucent included systems engineering, product management, and strategy development in the area of communications systems. He holds an M.S. degree in Computer Engineering from Florida Atlantic University and a B.S. degree in Electrical Engineering from The Technical Institute of Brno, Czech Republic. His e-mail is karel.rasovsky at intel.com.

Raj Sistla is a senior software engineeer in the Embedded IA Division (ICG/NPG) at Intel. His interests include modular servers and server mangement. He received his M.S. degree from the State University of New York at Buffalo and is currently working on his M.B.A from Indiana University. His e-mail is raj.sistla at intel.com.

Fabrics and Application Characteristics for AdvancedTCA* Architectures

Brian Peebles, Intel Communications Group, Intel Corporation
Chuck Narad, Intel Communications Group, Intel Corporation
Victoria Genovker, Intel Communications Group, Intel Corporation
Karel Rasovsky, Intel Communications Group, Intel Corporation
Jay Gilbert, Intel Communications Group, Intel Corporation

Index words: AdvancedTCA, aggregation, AS, Advanced Switching, ATCA systems, interoperable building blocks, PICMG, SERDES, switching fabric, topologies

ABSTRACT

After years of severe downturn in business activity, the communications industry is now embracing standards-based modular communications platforms (MCP) as a way to improve profitability while the market recovers. Eclipsing traditionally closed, proprietary communications systems, the adoption of MCPs promises Telecom Equipment Manufacturers (TEMs) lower development costs and a robust ecosystem of interoperable hardware and software building blocks, while at the same time making it easier to develop and run applications. It also promises Service Providers (SPs) smaller initial outlays and reduced Total Cost of Ownership (TCO). The MCP paradigm is rooted in several industry-wide standards initiatives and spans a broad ecosystem of building blocks including silicon, blades, chassis, backplane fabrics, operating systems, middleware, applications, and more.

The shift toward standardized hardware modularity began with Rack-Mounted Servers (RMS) that were optimized for compute-centric applications in enterprise networks and back-office systems. To accommodate the extreme requirements of central-office environments, however, a more flexible, reliable, modular, scalable, and higher performance approach was needed; thus, the concept of moving from proprietary systems to a standardized "bladed shelf" was born. Recently, the industry represented in the PCI Industrial Computer Manufacturers Group (PICMG) standardized a bladed form factor in a series of specifications known as AdvancedTCA[*] (ATCA).

Founded on the requirements of the communications infrastructure, PICMG 3.0 specifies the electro-mechanical, interconnect topology, and management aspects for building a modular, reconfigurable shelf. The large form factor of ATCA blades provides ample architectural headroom for next-generation silicon and applications. Switched backplanes, agnostic of fabric technologies (e.g., InfiniBand*, PCI Express*, Advanced Switching, StarFabric) facilitate extensive bandwidth scalability for a broad range of applications. The robust ATCA shelf management is based on the Intelligent Platform Management Interface (IPMI) 1.5 specification and it enables hot swapping, inventory information management, power distribution, and management facilities.

ATCA takes a holistic approach to High Availability (HA) and five 9s (99.999 %) reliability required in carrier-grade equipment through the integration of features for resiliency, redundancy, serviceability, and manageability. ATCA also lays a solid foundation for scalability of system performance and capacity. Application and control processors, network processing blades, Digital Signal Processing (DSP) blades, and storage blades can be hot-swapped on demand with profile-based interoperability, maintainability, and reliability. With its emphasis on low-cost manufacturing and the economies of merchant-market

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

volumes, ATCA-based platforms will lower the cost of network elements.

Advanced Switching (AS) is a new standard from the Arapahoe Working Group, which builds on the physical and link layers of PCI Express and defines a scalable, protocol-agnostic fabric featuring traffic differentiation to support QoS, congestion management, and support for easy management and HA solutions. AS is currently at Release Candidate 1.0. An emerging ecosystem of interoperable components makes AS an attractive candidate for the backplane fabric in modular communications equipment.

In this paper, we explain how MCPs are designed around standard modular building blocks: silicon, hardware modules, fabric management systems, operating systems, and middleware. In particular, we focus on the AS fabric that is designed to work in this space. Standardization and broad adoption of MCP platforms are linked to reliable interoperability. By focusing the industry on a limited subset of the myriad of options offered by ATCA, a robust ecosystem of interoperable components can develop and persist.

INTRODUCTION

As the boundary between enterprise and wide area networks begins to blur, a convergence of communications and computing platforms is underway. To facilitate this rapid evolution and the applications that are driving it requires not only significant computing power but a modular, scalable, and standardized approach to building hardware platforms that can support a broad spectrum of requirements for both compute and communications platforms. Born through intense industry cooperation, the recently ratified PCI Industrial Computer Manufacturers Group (PICMG) series of specifications also known as AdvancedTCA (ATCA) lays a foundation for building such a standard, carrier-grade infrastructure out of interoperable modular components.

One of the most critical aspects in enabling this technology is the ability for high-performance blades in communications and server chassis to communicate between and among each other moving vast quantities of data from blade to blade within a chassis (shelf). The switching fabric within a shelf enables the communication between blades and encompasses many switching technologies and numerous implementations, both standardized and proprietary.

We discuss the basic interconnect technologies available for ATCA systems including physical fabric interfaces and topologies as well as standard fabric technologies ranging from Ethernet through a new emerging standard referred to as Advanced Switching (AS).

AS picks up where Ethernet leaves off, providing lossless transfer, congestion management capabilities and traffic differentiation for QoS, bandwidth scalability, and other features important for communications backplane fabrics.

At the physical level, the ATCA switching architecture relies on a passive backplane with two physical interfaces, the Base interface and the Fabric interface. The Base interface defined in the PICMG 3.0 specification is 10/100/1000BASE-T Ethernet. BASE-T Ethernet provides backward compatibility with a large installed base of products and is suitable for many lower-bandwidth applications. The Fabric interface defined in the PICMG 3.1 specification utilizes the "SERDES" (Serializer/Deserializer) interfaces that can support much faster speeds and do away with the magnetics required for Ethernet fabrics. This specification supports many different topologies, from Dual Star to Full Mesh, making it a very flexible architecture. In a Dual Star fabric, each blade has a pair of redundant fabric interfaces, one connected to each of the two redundant centralized switches. In a Full Mesh system, each blade has a point-to-point connection to every other blade, and each blade has a switch fabric component to connect the on-blade interconnect to the backplane ports. Redundant paths can be supported through these switches for failover, and a Full Mesh eliminates the need for dedicated switch slots.

The Base interface is meant as a transitional interconnect that is identical to the one used in the PICMG 2.16 standard, for vendors porting products from that standard to ATCA. The system developer can decide how and when to use the Base and Fabric interconnects. Either can be used as a unified control and data fabric or the Base interface may be used for the control plane fabric with dual star, and a Fabric interface based on AS, InfiniBand, or XAUI might be used for a high-performance data plane fabric.

We describe practical ATCA switching fabric implementations, focusing primarily on the most useful topologies and switching technologies ranging from Ethernet to AS. We also include an example discussion of an AS hardware and software model architecture and give examples of communications applications.

ATCA SWITCH FABRIC AND CAPABILITIES

Fabric switching serves as the basis for interoperable blade-to-blade communications in a healthy ecosystem of interoperable building blocks used to develop systems that are based on the AdvancedTCA (ATCA) platform. When selecting the type of fabric switching to be utilized in a system, one must consider which protocol (e.g., Ethernet (10/100/1Gb), Fibre Channel, InfiniBand, StarFabric, Advanced Switching) and which topology (e.g., Star, Mesh) best suits the needs of the application. Additional considerations include standards-based platforms, commercial off-the-shelf silicon and blades, allowance for proprietary fabric interface implementations and protocols, and so on.

The PCI Industrial Computer Manufacturers Group (PICMG) specifications define a number of possible fabric implementations as shown in Table 1.

Fabric	Specification	Topology	Interface Type
Base Interface	PICMG 3.0	Dual Star	10/100/1000BASE-T Ethernet
Fabric Interface	PICMG 3.1	Dual Star	SERDES (e.g., 1000BASE-BX and 10 GbE XAUI) Ethernet
		Mesh/Full Mesh/Hybrids	
	PICMG 3.2	Dual Star	1X to 4X InfiniBand
		Mesh/Full Mesh/Hybrids	
	PICMG 3.3	Dual Star	1X to 4X StarFabric
	PICMG 3.4	Dual Star	1x-4x PCI Express and AS
		Mesh/Full Mesh/Hybrids	

Table 1: PICMG 3.0 specification fabric options

To enable a rich ecosystem of interoperable building blocks, it is necessary to select a limited set of profiles that will support a smooth transition over the development cycle of the technologies selected. Two such practical alternatives are Ethernet and Advanced Switching (AS), with our primary focus on the most useful topologies as depicted by the unshaded area in Table 1.

Fabric Physical Interfaces

The switching interfaces for ATCA backplanes utilize different physical interfaces as explained in the next two sections.

Base Interface

The Base interface defined in the PICMG specification is 10/100/1000BASE-T Ethernet. The Ethernet BASE-T interface employs four pairs of transmission lines (typically known as twisted-pairs in cables). The transmission lines on a backplane are implemented as pairs of copper traces whose characteristic impedance is the same as the twisted-pairs of a typical Category 5 Ethernet cable. An Ethernet port implements connections to a network or to a backlane switch port with a transformer (magnetics) to match the unbalanced transmit (output) and receive (input) of the Ethernet chip to the cable or backplane traces. The use of the magnetics is typically implemented to allow driving relatively long copper wires (or traces in the case of a backplane) while maintaining excellent signal integrity; that is, good balance and no DC offset. Use of the BASE-T interface provides backward compatibility with a large installed base of products, making it easier to quickly move functionality, previously provided in a proprietary system or a PICMG 2.16-compliant system, to an ATCA blade.

Fabric Interfaces and SERDES Interfaces

The Fabric interface defined in the PICMG 3.x subsidiary specifications utilizes up to eight pairs of transmission lines and support protocols such as Ethernet, InfiniBand, and Fibre Channel over a SERDES interface. The SERDES interface eliminates the requirement for magnetics on node blades and fabric switch blades that are used on the Base interfaces. The use of SERDES interfaces reduces blade real estate and power consumption because the length of copper trace pairs on a backplane is generally quite short compared to typical cable runs such as Category 5 Ethernet. SERDES interfaces can support much faster speeds than long cable runs and have been tested up to 3.125 Gbps with technology and roadmaps extending out to 12 Gbps.

Figure 1 illustrates a typical SERDES interface.

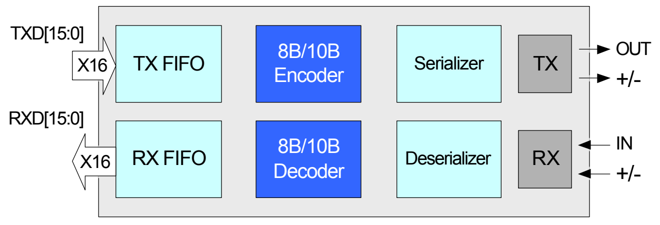


Figure 1: Block diagram of a typical SERDES interface

Differential signaling standards such as low-voltage differential signaling (LVDS) make it possible to transmit high-speed serial data at higher rates than conventional single-ended I/O standards.

Star and Mesh Topologies for AdvancedTCA

Figure 2 illustrates a typical ATCA backplane implementing a full mesh Fabric interface. Designed for an EIA 19" rack, this backplane can support up to 14 ATCA-compliant slots. The standard supports other topologies that are subsets of the mesh implementation such as Dual Star and Dual-Dual Star (two switches in each of the two switch slots) with implementations generally dependent on the platform's intended application, such as wireless network access or edge routing.

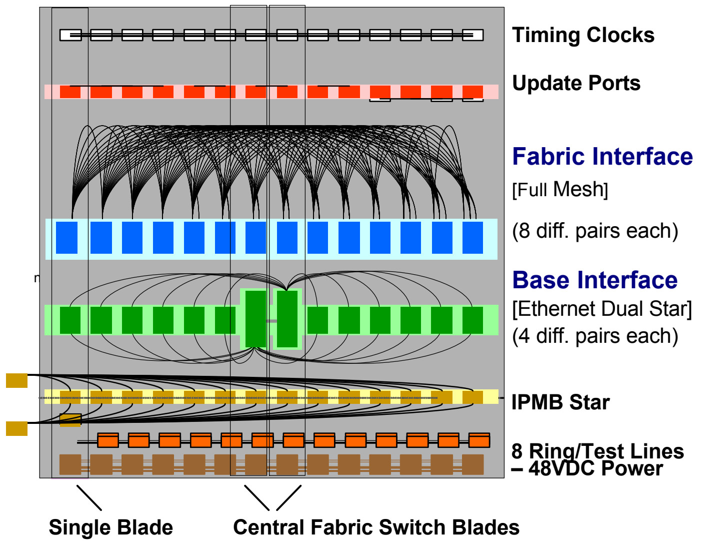


Figure 2: ATCA backplane with a mesh fabric interface

Backplane Fabric Bandwidth

The bandwidth of the fabric backplane is a function of the access aggregate bandwidth per port (or per node) and the total offered bandwidth of the backplane. The most strenuous I/O applications come from multi-service switches and edge routers where these applications may require a shelf full of OC48 interfaces or perhaps multiple OC192 and several OC48/12/3 interfaces to support an access distribution or fan-out. Wireless infrastructure applications currently require up to eight OC3s and two OC12s per shelf. Eventually, wireless applications will require more bandwidth as data-enabled handsets become more prevalent and drive infrastructure requirements.

A typical deployment might be two to four OC192 ports and eight to sixteen OC48 (or 32 to 64 OC12) ports. The presence of OC192 ports requires the access rate per port to be at least 10 Gbps plus some speed-up to account for backplane protocol overheads. It is obvious that when using 10 Gb Ethernet as the fabric, due to pin limitations it simply may not be possible to employ a dual star topology while supporting the 15-20 Gb/Sec needed per slot. However, as faster SERDES devices emerge along with more exotic backplane Printed Circuit Board (PCB) dielectric materials and connectors, which enable higher signaling rates, they will allow higher performance in simpler topologies.

Allowance for Proprietary Fabric Interface Implementations

Support for a proprietary fabric interface implementation may be needed for some systems in the near term. Since the ATCA backplane fabric interface is fabric agnostic, there are proprietary fabric implementations that can be implemented on an ATCA backplane provided they meet the fabric interface's electrical properties and support a SERDES interface.

E-Keying to Provide Safety

E-Keying is a form of shelf management software control that provides safety to both the shelf and the blades inserted into them. E-Keying prevents blades from inadvertently being enabled with the wrong capabilities by the Chassis Management Module (CMM) (e.g., incorrect power level) when initially plugged in to the chassis. E-Keying is controlled by the CMM with information derived from the Chassis Data Module (CDM), often called the CDM "FRU" (Field Replaceable Unit). E-Keying, along with other blade and chassis safety mechanisms, such as mechanical alignment pins, ensures that blades are properly placed within the chassis slot that is intended to support the operational requirements of that blade.

THE ADVANCED SWITCHING FABRIC

Gb Ethernet works as a fabric for many applications. More demanding applications require traffic differentiation and congestion management in order to implement Quality of Service (QoS). Support for data integrity, lossless transmission, fabric manageability, and other high-end features, plus the ability to scale bandwidth in reasonable increments, are all important features for many applications where simple over-provisioning of Ethernet bandwidth no longer suffices.

The Advanced Switching (AS) fabric is an evolution of the PCI Express fabric with which it shares the physical and link layer. AS goes beyond the load and store nature of PCI Express by providing a number of features that enable it to operate as both a data and control plane fabric for low-, medium- and high-end server and communications applications.

Among some of the features of AS are its self-routing nature, congestion management, scalability, multi-hosting and QoS.

AS uses the same 2.5 GHz SERDES using an 8B/10B block coding physical layer interface as PCI Express and InfiniBand. Each link is configurable in 1x, 2x, 4x, 8x and higher lane aggregations to provide considerable bandwidth scalability at 2 Gbps delivered per lane. The physical layer is isolated so that evolution to faster links is straightforward.

AS utilizes a link-level Credit-Based Flow Control (CBFC) scheme and has provisions for optional Explicit Congestion Notification (ECN) and Status-Based Flow Control (SBFC) as congestion management facilities.

AS provides for Virtual Channels (VC) as well as Traffic Classes (TC) to enable traffic isolation for QoS. AS also implements a Protocol Interface (PI) mechanism that allows easy upper-layer protocol encapsulation.

An AS fabric comprises end nodes and switch elements. A fabric might have one level of switching, or switch elements can be cascaded in a variety of topologies including Star and Full Mesh.

AS can be used as a light-weight-overhead fabric in and of itself or it can encapsulate other protocols and forward traffic transparently across the fabric.

AS traffic can be unicast or multicast and is self routing. Each packet contains a header that indicates the path the packet is to take from the source end node to the destination end node, and that identifies the payload type with a Protocol Interface (PI) number. In addition to the unicast header there is a "Path Building" header format and a separate multicast header.

Several PIs are reserved for management packets. All devices must support PI-0 and PI-4 and must be capable of generating PI-5, (although they are not required to respond to PI-5 should it be received). Most of the PIs are reserved for designation by the industry or are vendor specific. Table 2 gives a breakdown of the various PIs.

Table 2: Protocol interface values

PI Index	Protocol Interface	
0 (0:0) (0:8-126)	Path Building (Spanning Tree Generation) (Multicast) (Note that "0:x" notation indicates a PI-encapsulated PI.)	
1	Congestion Management	(Flow ID messaging)
2	Segmentation and Reassembly (SAR)	
3	Reserved for future AS Fabric Management Interfaces	
4	Device Management	
5	Event Reporting	
6-7	Reserved for future AS Fabric Management Interfaces	
8-95	Governing Industry (AS) SIG-defined PIs	
96-126	Vendor-defined PIs	
127	Invalid	

Turn-Based Routing

The AS fabric has a self-routing mechanism that is based on a technique called *turn-based routing*. The header contains three fields related to this mechanism: the *turn pointer*, the *turn pool,* and the *direction flag*. The turn pool contains the path from the source to the destination node. Each switch element (SE) along the path taken by the packet interprets a field in the turn pool (that SE's *turn value*) containing enough bits to select one of its ports; the width of that field varies from SE to SE. The turn pointer points to the start of the next field to interpret. After an SE has moved a packet from the ingress port to the egress port it moves the pointer by the number of bits it has consumed to point to the start of the turn value field for the next SE to use.

Rather than identifying a port by an explicit port number, path routing treats the ports in a switch element as a circular list and uses the turn value to indicate the relative position of the egress port to the ingress port. This mechanism eases path discovery and also simplifies the creation of reverse paths.

A path can be up to 31 bits in length.

As an example in the forward direction, if the turn pointer was pointing at bit<8> of the path and an SE has eight ports (three bits), then on egress from that SE, the turn pointer will point at bit<5> of the turn pool, which is the start of the next SE's turn value. Note that this mechanism allows for switches in multiple hops to have different port sizes, but the aggregate size of the switch ports combined in all hops is limited to the 31-bit size of the turn pool. If for example all switches had thirty-two ports (five bits), then a maximum of six switch hops could be employed in any one path before exhausting the turn pool.

If the direction flag is set, the packet is said to be moving in the "backwards" direction and the turn pool list is read from least to most significant bit. Backwards traversal of a path is used for example when sending a Backwards Explicit Congestion Notification packet, as explained later in the Congestion Management section of this paper.

If the direction flag is set to 0, the packet is moving in the forward direction and the turn pool is read from most to least significant bit of the valid field in the pool.

QoS in AS

Quality of Service (QoS) in AS is provided by a set of logical channels within a physical link known as Virtual Channels (VC). Each VC provides its own queue so that blocking in one VC doesn't cause blocking in another. Since each VC has independent packet ordering requirements, each VC can be scheduled without dependencies on the other VCs.

There can be as many as 20 VCs in a given link. A minimum of one and up to eight bypassable VCs (BVC) are allocated to both ordered and potentially bypassable unicast traffic, up to eight ordered VCs (OVC) are allocated to ordered-only unicast traffic and up to four are allocated to multicast traffic (MVC). BVCs and OVCs are further described below.

VCs can be used to establish connections between endpoints for control, management, and data.

Typically, when establishing a communications fabric, traffic will be segregated into service classes. These service classes correspond to the importance of the traffic or to delivery parameters such as QoS associated with that traffic. High-priority traffic, or traffic upon which the system (or the revenue-generating application that runs on the system) depends is considered "guaranteed" traffic. Guaranteed traffic must meet some set of delivery criteria in terms of overall latency and assurance that the packet is delivered. Less important traffic, such as real-time media may have low latency requirements, but can tolerably be undelivered in the event of congestion without too much degradation in quality. Some management and house-keeping traffic or some service traffic (e.g., web browsing) may be re-transmittable and have no real latency requirements. This traffic is considered "best effort" traffic in that it is transmitted when there is no other, more important traffic pending in an output queue.

The packet header indicates which of eight Traffic Classes (TC) the packet is to use to traverse the fabric. Each transmitting port maps that TC to a VC for that link.

The eight TCs are distributed across the supported VCs; when fewer than eight VCs are supported, then multiple TCs can map to a VC. The TC can be used to apply particular serving policies at any point (i.e., a switch) in the fabric. There is no provision in AS to have independent queuing mechanisms for TCs in the fabric, so different traffic classes sent over a common VC will share queues and can experience head of line blocking among those TCs. Traffic differentiation really requires independent VCs (queues) for each non-interfering class of traffic.

There are two types of Unicast Virtual Channels (VC) in AS: Ordered-Only (OVC) and Bypass Capable (BVC). The packet header identifies which type of VC the packet is to be enqueued on, an OVC or a BVC. An endpoint must support a minimum of one BVC. An OVC contains only one queue, while a BVC contains two queues: *ordered* and *bypass*. The packet header indicates if a packet destined for a BVC should be considered an ordered or bypassable packet.

Due to the nature of some load and store protocols (such as PCI Express "non-posted requests"), packets can get stuck in a queue pending sufficient credit. Packets that are sent to a BVC propagate through the ordered queue just like they would through an OVC; however, when they get to the head of the queue the packets are evaluated to see if they are bypassable. If it is marked bypassable, and there is insufficient credit to forward it, a packet is moved into the tail end of the bypass queue, where it waits until sufficient bypass credit exists for it to be forwarded. This allows ordered packets with sufficient credits to bypass stalled bypassable packets.

Congestion Management

There are several features (some mandatory and others optional) of AS that affect congestion management. The purpose of these mechanisms is to provide a robust fabric that will reliably forward data despite network irregularities. While some mechanisms merely preserve the operation of the fabric (at the expense of QoS), other mechanisms are designed to potentially prevent the fabric from becoming congested in the first place.

At the link layer, a Credit-Based Flow Control mechanism prevents packet loss by queuing packets until sufficient transfer credit is available. Credit information is exchanged between the two endpoints of every link. Credits are created per VC at the receive side of the link and communicated upstream to the transmit side of the link where they are accumulated. Credit is created at a particular destination queue when an amount of traffic has been forwarded or consumed at the egress side of that queue, thereby freeing up queue space.

The packet is transmitted only if there are enough credits accumulated for the particular VC queue in which the pending packet is queued. Once enough credits are accumulated, the transmit side of the link sends the packet, then deducts an amount of credit (for that VC) by the packet size of the transmitted packet.

While the CBFC ensures that no packets will be lost due to congestion, it tends to violate any service class agreements during times of congestion. To provide a more fair mechanism, and one that can help prevent congestion in the first place, Explicit Congestion Notification (ECN) and Status-Based Flow Control (SBFC) may be employed. Both of these mechanisms are optional in the AS fabric. The ECN mechanism works end-to-end in the fabric, whereas SBFC is more local to a particular link.

ECN is an optional threshold-based mechanism that monitors the downstream (egress) queues of a switch to detect when they reach a certain (tunable) threshold level, indicating that the downstream could be experiencing congestion. When this threshold is reached, the next packet is marked by setting its Forwards Explicit Congestion Notification (FECN) flag, and a Backwards Explicit Congestion Notification (BECN) message is sent by the switch back to the source. Upon receiving a BECN, the source should throttle back its packet stream and must respond to the switch with a Congestion Management Message (CMM) in acknowledgement.

Downstream switches seeing that the FECN bit is set on this packet will not send further BECN messages upstream to the source. The end point receiving a FECN-marked packet may choose to do something at an application level, but any such action is beyond the scope of the AS specification.

Optional SBFC employs special messages sent between link partners that are used to manage the transmit end of flows entering a congested queue at an egress port of downstream switch element. A target queue at the egress of the next-hop switch can throttle transmission from a specific upstream link partner queue by using transient or persistent XOFF and XON messages. The idea is to manage potential congestion conditions at a local level, before they impact the greater fabric.

With all three congestion management methods in place, the SBFC would attempt to predict congestion before it happens and alter the ordering of queued packets targeting different destination queues to prevent it. ECN would identify congestion as it is happening and notify the source so that it can prevent further congestion by altering its injection rate. CBFC would activate once congestion is already in progress and it can act to prevent packets from being lost, but does nothing to compensate for the congestion in the event that the source is still misbehaving and could cause congestion to spread through the fabric. Ideally the other mechanisms would prevent CBFC from ever kicking in.

If a packet is marked as perishable, then an SE can choose to discard it in the presence of congestion.

FABRIC MANAGEMENT IN ATCA USING EXAMPLES OF ADVANCED SWITCHING SOFTWARE ABSTRACTION LAYERS

AdvancedTCA (ATCA) features and capabilities mentioned in previous sections mostly concern themselves with ATCA hardware, while standardization of software features, most importantly fabric management in ATCA, is still off in the future. Although ATCA defines a very powerful infrastructure, the new technology's value will be fully realized only with intelligent software. In the rest of this section we explore the option of Fabric Management (mainly Fabric Discovery and Multicast) in ATCA by using the Advanced Switching (AS) software abstraction layers and algorithms.

Fabric Management Architecture for ATCA (Multi-Fabric Model)

To foster standardization and to realize the benefits of software commonality, a flexible Fabric Manager (FM) for ATCA provides an abstraction layer which as much as possible hides the details of the underlying fabric. Among services provided by a fabric manager are fabric discovery and management of multicast through the fabric

Distributed Fabric Discovery Algorithm

The very first step an FM has to do is discover the fabric it is managing. Fabric Discovery (FD) is one of the key software components of the fabric management suite. During FD, the FM records which devices are connected, collects information about each device in the fabric, and constructs a map of the fabric. There are several approaches to how the FM might collect the information about all the devices. The approach chosen in this paper is a fully distributed mechanism, meaning, at any given time, the FM might be collecting information about more than one device. For each device in the fabric, the FM takes the following steps:

- Locates the device (active ports analysis).
- Reads the device's capabilities (including writing fabric-specific information into certain capabilities).
- Reads tables referenced by these capabilities.
- Updates its own tables based on information read from the device's capabilities/tables.
- If all devices have been discovered, constructs the shortest path table between every pair of devices in the fabric.

AS devices provide data structures similar to PCI capability registers to describe supported functionality. The first 256 bytes of an AS device's configuration space are identical to a PCI device's configuration space, which categorize the device.

The unique set of features supported by a particular device can be extracted from a linked list of capabilities located in the device's configuration space and initialized by the device during power-up time. Each capability is distinguished by a unique Capability ID and provides an offset to the next capability in the list. An offset equal to 0 indicates that the end of that capabilities list has been reached.

The first device FM discovers is the switch to which it is connected. For each capability read, the FM does the following:

- Checks whether this capability references tables, and if so sends packets to read the tables.
- Checks whether it needs to update its tables based on information found in the capability.
- Sends a packet to read the next capability.

If all capabilities have been read for a particular device, and the device is a switch or a multi-ported endpoint, the FM sends out packets on all active ports of that device (except for the port through which the device itself has been discovered) to find new devices. This is the distributed nature of the algorithm. Instead of discovering devices one port at a time, the FM discovers devices on all active ports simultaneously.

Information collected by the FM about the devices includes the number of physical ports on a device, the status indicating which ports are active, events supported by a device, and more. If a device is an endpoint, then the FM also gathers information on which PIs that endpoint supports. If the device is a switch, the FM needs to read the switch's multicast support.

In order for an FM to distinguish between new and already discovered devices, the serial number of each device must be read. There are three cases that need to be considered when evaluating the serial numbers:

1. The serial number is all Fs:
   a. In this case, the FM has encountered a new device and its serial number has not been hard-wired by a manufacturer.
   b. The FM writes a fabric-unique serial number into that device and proceeds with the device's discovery.
2. The serial number is not all Fs and there is no record of it in FM's tables:
   a. In this case, the FM has encountered a new device. It makes a record of that device's serial number in FM's tables and proceeds with the device's discovery.
3. The serial number is not all Fs and there is a record of it in FM's tables:
   a. In this case, the FM has discovered a duplicate path to an already discovered device. The FM makes a note of it in its tables and *stops* discovering this device.

The FM keeps a list of the devices that are currently being discovered. When that list becomes empty, all reachable devices have been discovered. At this point, the FM calculates shortest paths between every pair of devices in the fabric, which will be used later for peer-to-peer communications, for example. Any duplicate paths found during discovery might be utilized during the run time of the fabric for fault resiliency or for traffic engineering to relieve chronic congestion. With a path-routed AS fabric, the path between any two end nodes is always unique. For efficiency and other reasons, some nodes might wish to run their own fabric discovery and collect information about the devices in the fabric.

Multicast Algorithm

One of the important features an FM should be able to manage during the run time of the fabric is updating the appropriate devices during the multicast group changes, i.e., a device has left or joined the group, or has changed its status (writer, listener, both) in the group. For the AS fabrics, the devices requiring updates are AS switches.

One approach is to keep the amount of the information required by the FM to perform the updates to a bare minimum. At the very least, the FM would have to maintain the number of paths going through the ingress and egress switch ports for a given multicast group. Each time a member joins or leaves a group, or changes its status in the group, all the FM needs to do is perform a simple check of its tables to determine if any switch's multicast table needs an update.

Also, since a mechanism to avoid looping in multicast is not included in AS, one of the software solutions is to build a Spanning Tree of the fabric and use that Spanning Tree for the shortest paths between the devices.

Combining the innovations of ATCA with sound software support will enable a smooth convergence between the communications and computing platforms.

APPLICATION EXAMPLES UTILIZING ATCA

There are two primary architectural models for AdvancedTCA (ATCA): server and communications. In the communications architecture, there are two models: transit and transformation. Each of these models has unique attributes that affect the usage and performance of the backplane fabric.

Server Applications

The server model consists of a chassis of single blade computers, each operating as a self-contained server. These servers use the backplane fabric as an extension of the Local Area Network (LAN) to which the ATCA chassis is attached. Figure 3 illustrates this concept. Typically, only a switch blade separates the LAN from the backplane, allowing the backplane to operate as a subnet or as part of the actual LAN. Switch blades are shown in pairs for redundancy. Redundancy can be served in many forms: (1) hot spare with all traffic identical across both switches, (2) warm spare where traffic only passes through the active switch, and (3) load balanced where different loads pass over each switch, but managed so that if one fails, the remaining switch can handle all traffic.

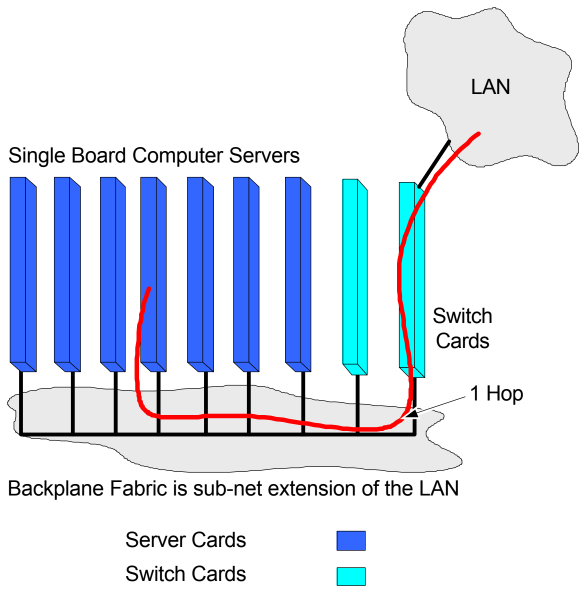

LAN
Single Board Computer Servers
Switch Cards
1 Hop
Backplane Fabric is sub-net extension of the LAN
Server Cards
Switch Cards

Figure 3: Server model application for ATCA

The backplane can also be used as a Storage Area Network (SAN) access media in this application. Signal flow originates and terminates at each individual server blade, flowing through the local switch blade to and from the LAN. The bandwidth demand of the backplane scales with the number of servers in the chassis.

For these applications, Ethernet or InfiniBand tend to work best, as such fabrics can be made homogeneous with an external LAN of the same type. Advanced Switching (AS) can also be used in this application, but it requires additional functionality to bridge between the AS backplane fabric and the external LAN to make the AS fabric appear as a subnet of the LAN.

Transit Applications

Transit applications operate as a core router or switch at the edge or on an internal core network. In such applications, the ATCA chassis sits between two or more Wide Area Networks (WANs), Metropolitan Area Networks (MANs), or LANs–in some combination. Each line card (blade) functions as an access to and from one of those external networks. The signal flow requires a hash or table look-up that is local to the ingress line card that subsequently forwards the packet to the switch, which then forwards the packet to the egress line card. Additional management packets propagate the table updates, and some amount of protocol translation may occur between the ingress network and the egress network. If the application encapsulates the ingress traffic, then there will be a net increase in overhead and bandwidth across the backplane. Figure 4 illustrates an example configuration of the transit application.

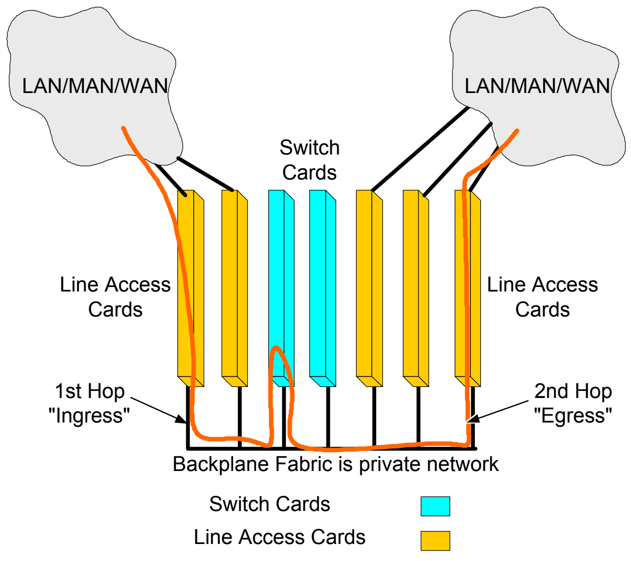


Figure 4: Transit application in ATCA

In addition to the data traffic flowing between the external networks, the shelf also generates internal control information (e.g., state, management, table updates) that needs to be passed between control endpoints in the fabric. This data can be merged onto the common fabric interconnect or can be routed separately via the base interconnect.

The line cards can be symmetrical in terms of the access rates to the various external networks, or they can be asymmetrical where lower rate line interfaces on one network may aggregate up to a higher rate interface on another network. All such traffic is peer-to-peer with the routing path determining what traffic aggregates to what egress line card.

Performance for these applications is very demanding, requiring very high bandwidth accompanied with flow control and backpressure mechanisms in order to keep the throughput to a maximum level by avoiding head of line blocking in the fabric.

In order to function well, this application requires the more advanced features of AS, such as flow control and Quality of Service (QoS) support, which Ethernet lacks.

Transformation Applications

Examples of a transformation application consist of wireless infrastructure or enterprise content processing equipment. Content processing applications include Virtual Private Networks (VPN), firewalls, virus scanning, intrusion detection, and others. A transformation application functions as edge or access equipment that sits between two or more networks much like the transit application. Wireless applications have a relatively constant traffic arrival rate (not as bursty as core network applications). Some content processing applications collect long windows of data using a SAN and conduct trend analysis or pattern matching on the incoming traffic, which can be very time consuming.

Packets are processed in blades within the chassis. The signal flow typically requires the line (network access) cards to terminate the session, normalize the ingress network protocols, and load balance the ingress traffic across one of several processing blade destinations. The normalization of the incoming traffic translates the protocol of the ingress network to a protocol that is optimal for routing within the backplane. This may involve a simple encapsulation, but typically the ingress network protocol is replaced with an internal protocol. This action can actually reduce the net overhead of the incoming traffic, leading to an efficiency improvement on the backplane fabric.

Processing blades typically terminate the session at some level or process the upper-layer protocols before forwarding the traffic to the destination line access blade on the egress side. The egress line card terminates the internal session and specializes the protocol to match the external network. This flow is illustrated in Figure 5. Notice that the local switch handles two passes of the traffic between ingress and egress. The protocol processing blades can consume or generate traffic or they can inspect or modify the upper layers of the packets they process. This type of operation distinguishes it from the transit applications in that system end-to-end latency is typically much greater due to the processing, and the ingress and egress flows are not necessarily symmetric. This leads to a more complex traffic model, but one that typically does not suffer from the congestion issues faced by transit systems at the core network.

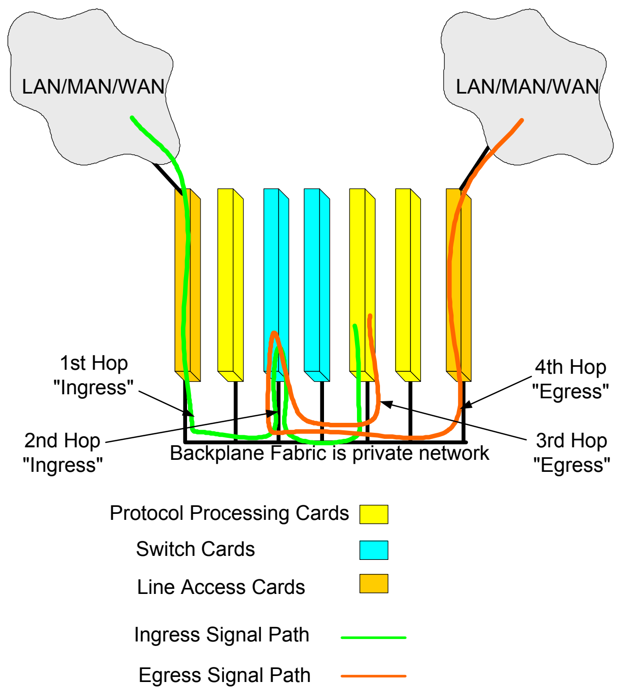


Figure 5: Transformation applications in ATCA

The ingress line card fabric access bandwidth is typically many times the capacity of the protocol processing blades. Therefore, the balance between processing blades and line cards is typically many to one. Fabric performance is relatively high, using per-flow management and a speed-up ratio of 2 or 3 to 1 over the ingress line rates. Ingress and egress line cards might not be symmetric. Sometimes, for example, several OC3 lines aggregate into a fewer number of OC12 lines.

This application can make use of either Ethernet or AS (or even InfiniBand) equally well. Since ingress traffic is relatively non-bursty and load balanced at the line interface, statistical blocking is not a major issue; thus, the more advanced flow and QoS features of AS are not required. However, the link granularity of AS in this space is highly beneficial as this class of applications requires finer increments of fabric bandwidth to better match the wide range of expected performance. Furthermore, the switching function is a simple Layer 2 operation that ideally would take advantage of virtual output queuing to minimize the amount of speed-up that is required.

Lastly, it bears mentioning that ATCA has a fair amount of bandwidth-growth potential. Although 2.5 GHz SERDES are identified today, we can project the bandwidths that may be achieved in the future given better SERDES rates and using Mesh topologies. Table 3 illustrates this point.

Table 3: Projected bandwidths with faster SERDES and mesh topologies

Fabric	Per Node Access BW for Redundant Star	Per Node Access BW for Mesh
Standard Gigabit Ethernet (1 Gbps)	1 Gbps	14 Gbps
AS, IB or Ethernet over SERDES w/ link aggregation		
2x 2.5 Gbps SERDES	4 Gbps	56 Gbps
4x 2.5 Gbps SERDES	8 Gbps	112 Gbps
4x 6.25 Gbps SERDES	20 Gbps	280 Gbps

CONCLUSIONS

The AdvancedTCA (ATCA) standard provides a highly flexible and scalable architecture for fabrics in its backplane. While still considered “new” by industry standards, it is quickly gaining in popularity because it was designed from the beginning with the server and telecommunications equipment markets in mind.

The new AS fabric definition provides a low-overhead, scalable fabric that meets the demands of high-end transit applications while not being overly complicated for lower performance, transformation, and server applications.

In addition, it can be seen that fabric management in ATCA is a fairly straight-forward process, especially when using the AS fabric.

With a wide range of fabrics to choose from, a vendor can apply ATCA across numerous applications in both the server and telecommunications markets. Common interoperable profiles might use just the Base or Fabric interface as a unified data-plus-control fabric, or might use Base for a control fabric and Fabric with InfiniBand or AS as a high-performance data plane fabric. While Ethernet, InfiniBand, and AS are all defined to operate in ATCA, we find that AS is particularly well suited to communications as it provides the greatest flexibility and desirable fabric features; AS was designed predominantly as a backplane and chip-to-chip interconnect, as opposed to a LAN fabric that was extended to the backplane. As communications and compute continue to converge, we expect that the needs of the more demanding communications applications will drive the choice of the shared backplane fabric.

ACKNOWLEDGMENTS

Narjala Bhasker of CIG’s Technology Office also contributed to this paper.

REFERENCES

[1]Arapahoe Working Group, Advanced Switching Core Architecture Specification, Release Candidate 1.0, October 2003.

[2] PICMG, PICMG 3.0 AdvancedTCA^{TM} Base Specification, Revision 1.0, December 30, 2002.

[3]PICMG, PICMG 3.1 Ethernet/Fibre Channel for AdvancedTCA^{TM} Systems, Revision 1.0, January 22, 2003.

[4] PICMG, PICMG 3.2 InfiniBand for AdvancedTCA^{TM} Systems, Revision 1.0, January 22, 2003.

[5] PICMG, PICMG 3.4 PCI Express and AS for AdvancedTCA^{TM} Systems, Draft Revision 0.8, March 19, 2003.

[6] IPMI-Intelligent Platform Management Interface Specification, V1.5, Revision 1.1, February 20, 2002.

AUTHORS' BIOGRAPHIES

Brian Peebles is a principal architect in the Network Building Blocks Division of CIG with nearly 20 years of experience in the communications field. Brian's range of expertise includes radio, telephony, and data communications. His current activities focus on future wireless communications architectures, communications fabrics, and system-level hardware and software architecture and design. Brian has M.S.E.E./B.S.E.E degrees from New Jersey Institute of Technology. His e-mail is Brian.Peebles at intel.com.

Chuck Narad is a principal system architect in Intel's Communications Infrastructure Group Technology Office. His technical interests and background span workstation, server, and supercomputer systems architecture, CPU design, hardware/software interfaces, interconnects, network processing, and network processors. He has a B.S.E.C.E. degree and an M.S.E.C.E. degree from UC Santa Barbara, holds 17 patents and has over a dozen more pending. His e-mail is Chuck.Narad at intel.com.

Victoria Genovker is a software engineer in the Embedded IA Division of the Network Processing Group in CIG. She is a member of the Advanced Switching team which is simulating AS fabrics in software to validate the Advanced Switching specification. Victoria holds a B.S. degree in Computer Science from the University of Arizona and is finishing her M.S. degree in Computer Science from the National Technological University. Her e-mail is Victoria.V.Genovker at intel.com.

Karel Rasovsky is a technical marketing engineer in Intel's Marketing and Platform Programs. He is currently responsible for introducing Modular Communications Platforms to the market. Prior to Intel, he was a product marketing manager at Lucent Technologies, focusing on converged voice and data solutions for next-generation networks. His other roles at Lucent included systems engineering, product management, and strategy development in the area of communications systems. He holds an M.S. degree in Computer Engineering from Florida Atlantic University, and a B.S. degree in Electrical Engineering from Technical Institute of Brno, Czech Republic. His e-mail is Karel.Rasovsky at intel.com

Jay Gilbert is a senior technical Marketing Manager within Intel's Communications Infrastructure Group. He is responsible for standards development, customer training, and technology evangelism in support of Intel's AdvancedTCA communications infrastructure development efforts. He has been with Intel Corporation for more than 12 years and holds a B.S.E.E degree from the Oregon Institute of Technology and an MBA from Portland State University. His e-mail is Jay.gilbert at intel.com

AdvancedTCA*/CGL Advantage: HA and TCO

Sharad Garg, Intel Communications Group, Intel Corporation
Raj Sistla, Intel Communications Group, Intel Corporation
Ramesh Caushik, Software and Solutions Group, Intel Corporation
Julie Fleischer, Software and Solutions Group, Intel Corporation
Rusty Lynch, Software and Solutions Group, Intel Corporation

Index words: Advanced Telecom Computing Architecture (AdvancedTCA), high availability (HA), Total Cost of Ownership (TCO), carrier grade, Linux, Carrier Grade Linux (CGL), Chassis Management Module (CMM), Telecom Equipment Manufacturer (TEM), Telco, five 9s

ABSTRACT

Carrier-grade systems in today's telecom market have high availability needs; however, current market conditions are forcing telecom equipment providers to look for lower-cost alternatives. In this paper, we discuss Advanced Telecom Computing Architecture (AdvancedTCA[*]), Carrier-Grade Linux[*] (CGL), and how these standards meet the requirements of carrier-grade equipment. We also describe how both of these standards can help increase high availability (HA) and reduce the total cost of ownership (TCO) in carrier-grade systems.

We define AdvancedTCA and its relationship to HA and reduced TCO. We also describe AdvancedTCA shelf management and show how AdvancedTCA hardware can enable HA middleware to perform HA functions. When discussing CGL, we focus on CGL requirements related to AdvancedTCA that increase HA and reduce TCO, as well as those that reduce TCO in general.

INTRODUCTION

In the current telecommunication climate there is a strong need to introduce increasingly complex voice and data services and at the same time reduce capital and operational expenditure. Next-generation telecommunication applications and carrier-grade equipment that provide these services have stringent availability (99.999% or greater, sometimes referred to as the five 9s), serviceability, and scalability requirements. In addition they need to have a low Total Cost of Ownership (TCO). Carrier-grade systems were until now built around proprietary architectures that led to unacceptably high TCO for these custom-designed solutions. Alternative architectures that are based on open, standards-based communication infrastructure platforms and hardware and software building blocks help provide the needed high availability (HA) and performance at a reduced TCO. AdvancedTCA is a specification that defines such an open modular architecture. It defines a new board and chassis form factor, and an interconnect fabric and management schema that is tailored to meet the HA needs of next-generation converged applications. AdvancedTCA-based hardware requires management software, a carrier-grade operating system and HA middleware to provide the total solution. AdvancedTCA, a standard proposed by the PCI Industrial Computers Manufacturers Group (PICMG), defines carrier-grade hardware in an open, standards-based way. The AdvancedTCA standard also enables centralized, scalable, out-of-band management in a high-density modular chassis by incorporating the concept of a shelf manager that acts as the central management authority in the system. In this paper, we discuss the Intel® NetStructure™ AdvancedTCA Chassis Management Module, an implementation of this concept.

The CGL Working Group of the Open Source Development Labs (OSDL) has defined a series of Linux requirements [9] that enhance Linux [10] for carrier-grade systems.

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

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

In this paper, we discuss AdvancedTCA, how it combines with carrier-grade Linux operating systems (OS), HA middleware, and the management software to deliver on the promise of HA with reduced TCO. In addition, this paper describes how CGL requirements enable the HA needed for carrier-grade systems as well as how the requirements enable an open, standards-based solution that reduces TCO.

ADVANCEDTCA

Overview

The Advanced Telecom Computing Architecture (AdvancedTCA) is a specification targeted at the next generation of telecommunications equipment. It provides the high-availability (HA) requirements needed for carrier-grade equipment, and its open, standards-based approach enables a lower Total Cost of Ownership (TCO) for Telecom Equipment Manufacturers (TEMs).

AdvancedTCA Advantages

The AdvancedTCA specification defines carrier-grade communications equipment that will be highly available and can have a lower TCO than proprietary solutions. The key hardware-related advantages of the AdvancedTCA-based solutions are listed below.

Standards Based

The advent of rugged, standards-based architectures resulting from the AdvancedTCA standard is expected to steer the market away from proprietary designs. These architectures allow TEMs to choose the best components for the system and can help decrease cost as well as time to market.

Adaptable Interconnections

The AdvancedTCA interconnect framework is comprised of the physical connector used to mate boards and backplanes, the mapping of interconnections to that connector, and the routing of those signals among boards across the backplane. The AdvancedTCA supports five different communication interconnects that can be used for control, data, and management traffic. AdvancedTCA does not mandate any specific interconnect technology (e.g., 10Gbe, StarFabric, InfiniBand, PCI Express, etc.) and is capable of supporting all of them due to the way in which the specification deals with just differential pairs and not the protocols/technologies. This provides a great deal of flexibility and protection for the future and reduces the TCO.

Hot Swap Ability

In AdvancedTCA, almost all the entities are required to be hot swappable. This ensures operation on the order of 99.999% since a hot swap of one component would not interfere with the payload operation of another component.

Built-in Manageability Support

Downtimes of less than five minutes a year is the norm for these mission-critical systems for telecom applications. As such it is imperative that these systems be constantly monitored and managed to anticipate and prevent failures. It is also important that the system management mechanisms do not interfere with the operation of the payload: i.e., out-of-band management is essential.

AdvancedTCA devotes a great deal of attention to system management. About 30% of the specification discusses the technology and the requirements for achieving system management in AdvancedTCA systems. AdvancedTCA uses the Intelligent Platform Management Interface (IPMI) v1.5-based protocol for management and extends it in multiple ways to achieve all the requirements. The physical medium for management transport across the backplane is a two-wire serial bus capable of carrying Intelligent Platform Management Bus (IPMB) messages. AdvancedTCA mandates that every entity that connects to a management bus shall implement a management Intelligent Platform Management (IPM) controller to communicate via the IPMI. Also, to achieve HA, the specification mandates that the management bus be paired up with a redundant IPMB. These requirements go a long way to ensure a standards-based management methodology, something that was not proposed by earlier standards and something that increases Mean Time Before Failure (MTBF) and reduces costs in the process.

SHELF MANAGEMENT IN ADVANCEDTCA

The importance assigned to shelf management increased greatly between Compact PCI and AdvancedTCA. The AdvancedTCA specification mandated the implementation of a shelf manager in the shelf (chassis) as a central management authority in the shelf. The purpose of shelf management is to monitor, control, and ensure proper operation of boards and other components by maintaining a health counter that reports anomalies and takes corrective actions when required. The shelf manager is also capable of retrieving inventory information from the shelf components.

In AdvancedTCA there are multiple components of the shelf management system.

- There are individual management controllers on each Field Replaceable Unit (FRU) in the system that monitor the system's operation and health.
- The shelf manager is the central management authority in the shelf and communicates to the other controllers via the Intelligent Platform Management Interface (IPMI).
- There is a higher-level external system manager that is in charge of managing multiple shelves within the Telco office.

The shelf manager in AdvancedTCA performs the following major functions for the shelf components:

- Power management
- Hot-swap management
- Cooling management
- Electronic Key (E-key) management
- Inventory management

AdvancedTCA introduces the notion of "Intelligent FRUs." An intelligent FRU is one that contains an IPM controller that implements the IPMI protocol and that is capable of interfacing to the Intelligent Platform Management Bus (IPMB). Each intelligent FRU must negotiate power-usage needs with the shelf manager before it can be powered on. The shelf manager manages the power and cooling resources in the shelf at all times to ensure that the headroom is not exceeded. Also, boards must report their backplane interconnection types before any of them are enabled. This last functionality, known as E-keying, replaces the legacy model of using mechanical keys on the backplane to prevent incompatible interconnections.

The Intel Chassis Management Module

Since shelf management is such an important piece of AdvancedTCA, a robust solution to shelf management is key to a full AdvancedTCA solution. The Intel NetStructure Chassis Management Module (CMM) is one such solution for AdvancedTCA shelf management. It provides centralized, out-of-band management in a high-density modular chassis that is compliant with PCI Industrial Computers Manufacturers Group (PICMG) standards. It implements a radial IPMB topology whereby each intelligent FRU is connected to the shelf manager over dedicated, redundant IPMB links and has eventing/alarming capabilities for up to 16 node and/or fabric slots as well as for system power entry modules and fan trays. This ensures greater throughput, less chance of error, and greater security. In addition, the active CMM may be paired with a backup CMM for redundant use in high-availability (HA) applications.

The CMM is capable of controlling, sequencing, and managing power to the FRUs in the chassis and maintaining a power budget. It manages the interconnections between the computing boards in the chassis ensuring that only boards using similar protocols are allowed to communicate over the backplane. The CMM maintains an inventory of the entire chassis to allow for easy field replacement of a module. The CMM employs different mechanisms to monitor and alarm, based on health changing events in the chassis. The adaptable designs of both the hardware and the software of the CMM enable Original Equipment Manufacturers (OEMs) and Telecom Equipment Manufacturers (TEMs) to build application-specific, standards-based modular platforms and extensions to the management software stack using the Intel CMM.

Given the critical nature of the applications using these modular platforms, it is unacceptable to have a single point of failure anywhere in the chassis. Since the CMM is the central management authority in the system, it is vital that there be a redundant CMM for HA failover.

The goals of redundancy between the CMM are twofold:

(1) Ensuring a seamless failover to the redundant CMM in cases of emergencies thereby preserving the notion of a single management authority in the system.

(2) Synchronization of the management information between the CMMs to maintain accuracy in cases of failover. The two CMMs operate in an active-standby model. The following components of the information are kept in sync between the CMMs:

  - IPMB state information
  - System event logs
  - Health information
  - CMM configuration information
  - User names and passwords
  - Power and hot-swap information
  - Cooling parameters
  - E-keying information

The physical media used for achieving the synchronization are the IPMB and the Ethernet. Two redundant and dedicated IPMBs connect the two CMMs. Each CMM has two network ports that can be configured to provide an Ethernet connection between the two CMMs.

The active CMM will failover to the standby when it senses that it is not capable of acting as the shelf manager and the standby is better able to do so. When a failover occurs, the standby takes over seamlessly to ensure active shelf management. This redundant operation increases the HA of the entire shelf.

HIGH-AVAILABILITY MIDDLEWARE AND ADVANCEDTCA

Configuration and cluster management middleware is the key controlling entity in a high-availability (HA) configuration and is implemented by HA middleware. Figure 1 shows the software stack and HA middleware that run on an AdvancedTCA blade. It maintains a system model of the components that comprise the cluster, defines how faults are detected in the cluster, and what action should be taken as a result. When failures occur it takes the appropriate actions to reconfigure the cluster and notify and/or restart affected parts of the application. The HA cluster configuration and management must be able to operate in a heterogeneous cluster. The member systems in the cluster may be of different types with different operating environments, and may have HA management middleware from different suppliers. The system model, the message protocols between cluster members, and application APIs must allow proper heterogeneous interoperability.

In this section, we discuss HA middleware requirements and show how HA middleware achieves application failover, including the fault management cycle. Then, we describe the hardware features of an AdvancedTCA platform that can assist HA middleware to achieve fast failover.

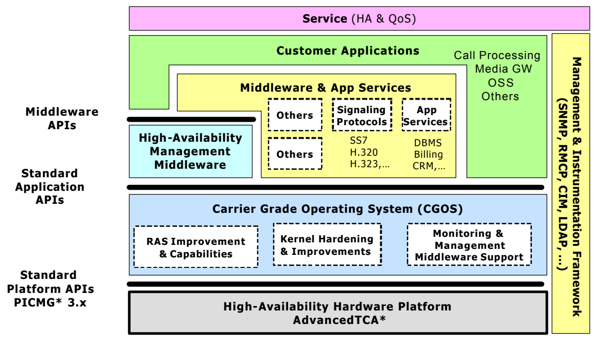

Service (HA & QoS)
Customer Applications
Call Processing
Media GW
OSS
Others
Middleware & App Services
Others
Signaling Protocols
App Services
Others
SS7
H.320
H.323,...
DBMS
Billing
CRM,...
Middleware APIs
High-Availability Management Middleware
Standard Application APIs
Carrier Grade Operating System (CGOS)
RAS Improvement & Capabilities
Kernel Hardening & Improvements
Monitoring & Management Middleware Support
Standard Platform APIs PICMG* 3.x
High-Availability Hardware Platform AdvancedTCA*
Management & Instrumentation Framework (SNMP, RMCP, CIM, LDAP, ...)

Figure 1: Software stack running on an AdvancedTCA blade

Telco HA Middleware Requirements

An application failover solution in a carrier-grade, HA environment must meet the following criteria:

(1) *Efficient and fast*. The management software must be optimized for speed and efficiency. Given the stringent performance and availability requirements of telecom equipment providers, it is important that the management software itself carries a low overhead [1], [2].

(2) *Self-managing*. The entire HA management system must be self-managing and self-reliant, and the failover cycle must operate automatically in real-time without need of human intervention. Self-management decisions are usually based on policies.

The management system must monitor and collect data about all critical system resources, including hardware, software, the operating system, and applications. Availability management software therefore must include an interface to these resources. System faults must be quickly detected, and fault data must be provided to an availability management service so that the service can quickly and appropriately respond by initiating actions that reconfigure the status and functioning of the resources as needed to maintain service.

(3) *Configuration and maintenance of system-wide state model*. Service availability management requires a system-wide model that can represent all managed resources in the system, changing information and the intricacies of each resource's dependencies and interdependencies. The management software needs this information in order to make quick and appropriate reconfigurations when necessary.

(4) *Automatic data check-pointing*. Redundancy is the key to HA. To provide continuously available services, the current application data and state must be maintained in a hot standby redundant resource. The HA management middleware must continuously copy ongoing transaction and application state data to a hot standby resource for a fast failover.

(5) *Fault detection and diagnosis*. The HA management middleware must detect and analyze errors, and it must initiate an appropriate recovery action including switchover to a standby resource and send a message.

(6) *Fast policy-based recovery*. The corrective action taken by the HA management middleware should be based on an application-specific policy. These actions can cover a wide range of activities from restart of a failed application, to failover, to a

standby hardware card. The recovery process is often multi-step in that several actions must be taken in a prescribed order. In some cases, the recovery process is multi-tiered in that, if a particular action does not recover the system, some alternative action must be taken.

(7) *Dynamic management of system component and dependencies.* Cluster-aware availability management is responsible for initiating the actions and orchestrating the role assignments that maintain continuous availability. Availability management also controls service groups (the collections of managed objects in redundancy relationships such as 2N or N+1) and makes role assignments (active, standby, spare) within them. Availability management is responsible for the ongoing verification of component health via heartbeating protocols, for overseeing the check-pointing of data to redundant resources, for executing the switchover to redundant resources, and for dynamically reconfiguring the system.

(8) *Administrative access and control:* Service availability middleware must provide a set of services for the ongoing maintenance and management of the system.

Fault Management Cycle

As just described in a previous section, fault detection and diagnosis, and a fast recovery are key requirements for a fast failover solution. The fault management cycle encompasses these steps, starting with the detection of a fault and ending with the recovery of the system [8]. A complete fault management cycle is described below, and the tasks a HA middleware solution should perform to ensure HA and a fast failover are also described.

Detection

The HA middleware will identify an undesirable condition by checking the health of every entity on a blade such as applications, the OS running on the blade, and hardware entities such as network port or memory faults, etc. The Intelligent Platform Management (IPM) device on each blade monitors several hardware/physical entities such as temperature and processor on that blade. In the case of hardware faults, the IPM device logs the events and faults in its local system event log (SEL) and informs the shelf manager. There can also be watchdog timers that might be monitoring some events, and in cases of time out, will inform the IPM device about such events. The HA middleware can access Intelligent Platform Management Controller (IPMC) event logs such as rising temperature, or a failed component to find out about such events, thus avoiding going through layers of software.

Diagnosis

The HA middleware analyzes faults to determine their nature and location. It also performs root-cause analysis to identify the originating fault.

Isolation

The HA middleware contains the problem to ensure that a fault does not cause a system failure.

Recovery

To recover the system, the HA middleware does the following:

- Restores system to expected behavior.
- Performs policy-based actions such as
  - restarting a failed application or
  - switching over to a standby node.
- The shelf manager and the IPM device can assist by, for example, rebooting or restarting a node.

Repair

The HA middleware repairs the system by doing the following:

- Restoring a system to full capability including all redundancy.
- Replacing hardware or software components.

Hardware Assistance by AdvancedTCA Components to a Fast Failover

AdvancedTCA can be used to achieve a fast failover with a fast fault management cycle process. The AdvancedTCA components listed below are key pieces in a fast failover. These pieces are shown in Figure 2 below [5].

(1) *Update channel.* This low-latency board-board channel can be used to check the heartbeat of the board. The HA middleware can use this channel to synchronize application state and data with a redundant application on the standby node. The Switch Fabric Bypass interface of the update port can be used for checkpoint application data and state information to a standby application running on another board, thus bypassing the high-latency switching fabric [3], [4], [5].

(2) *Shelf manager.* The shelf manager can use the health data provided the HA middleware to take

corrective action including generating alerts and e-mails, and reconfiguring a warm standby board.

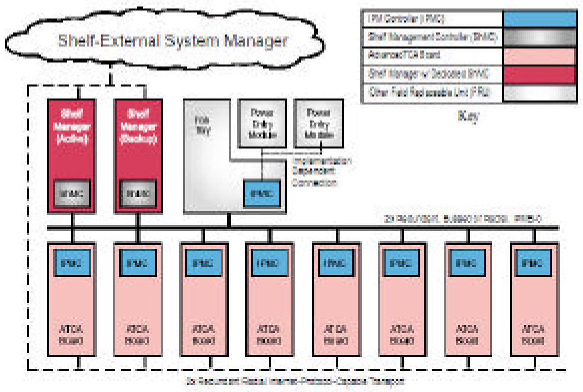

Shelf-External System Manager
IPM Controller (IPMC)
Shelf Management Controller (ShMC)
AdvancedTCA Board
Shelf Manager w/ Dedicated ShMC
Other Field Replaceable Unit (FRU)
Key
Shelf Manager (Active)
Shelf Manager (Backup)
ShMC
Fan Tray
Power Entry Module
Power Entry Module
Implementation Dependent Connection
IPMC
ATCA BOARD

Figure 2: Key components of a fast failover

CARRIER-GRADE LINUX

Along with hardware and system management, robust operating systems and middleware are imperative to achieving a truly high-availability (HA) system. Carrier-Grade Linux (CGL) addresses the issues of HA and total cost of ownership (TCO) at the operating system and middleware level. The Open Source Development Labs (OSDL) CGL Working Group is the standards group that produces the CGL requirements, which define the additions to Linux that are needed by the telecom industry. CGL requirements ensure that the availability and service response characteristics needed in a carrier environment are fulfilled, thus ensuring a system that is truly HA. Subsequent sections describe how CGL enhances platform availability and reduces TCO as well as how CGL supports HA middleware services, such as those standardized by the Service Availability Forum (SAF).

Carrier-Grade Linux and HA

The OSDL Carrier-Grade Linux Working Group took into account the special considerations of AdvancedTCA when developing the CGL requirements. Because of this, the CGL requirements outline several key areas of operating system support for AdvancedTCA, enabling solution providers to create HA solutions on top of CGL-conforming operating systems running on AdvancedTCA blades.

Carrier-Grade Linux Support for Platform HA

This section describes some platform-specific strategies aimed at increasing HA and the support the OS provides to enable these strategies.

Redundancy Support

Node boards that are used in AdvancedTCA-based equipment will typically employ device-level redundancy to eliminate single points of failure on the board. Such redundancy, in combination with the HA features in the AdvancedTCA chassis and other associated equipment (like the fabric/switch boards) and OS and middleware support, ensure the HA of specific functionalities. Two key functionalities that we address in this section are (1) Ethernet connectivity between and to the nodes in the chassis and (2) disk data storage.

Ethernet Connectivity

To ensure HA Ethernet connectivity between nodes in an AdvancedTCA system, the node boards will typically include redundant Network Interface Controllers (NICs). These redundant NICs are combined to appear as a single virtual Ethernet interface by using the technique of "Ethernet bonding." When one of the NICs in the bonded set fails, the other NIC (in the case of two bonded NICs) takes over transparently, thus preventing a single point of failure. A bonding driver combined with a system-level configuration facilitate Ethernet bonding. Support for Ethernet bonding is a key CGL requirement. Examples of conforming bonding driver implementations are the bonding driver developed by the SourceForge "Channel Bonding" project [16] and the Intel Advanced Networking Services Driver (iANS).

Bonding drivers can set up the bonded NICs to operate in a variety of modes ranging from the simple link failover mode where at any time only one NIC is active, to "link aggregation" modes where all the bonded NICs transmit and receive simultaneously, to varying degrees.

Disk Data Storage

To achieve HA of 99.999% or more, AdvancedTCA systems need to ensure that disk data corruption or disk failures do not lead to system downtime. Employing a Redundant Array of Inexpensive Disks (RAID) system is one way to meet this requirement. The OSDL CGL requires software RAID 1 [17] along with other user-space RAID tools (Disk Mirroring) support, which essentially provide for the maintenance of duplicate sets of all data on separate disk drives. In addition, the RAID 1 support will also allow booting off a mirrored drive if the other disk drive fails. In addition to mirroring the disk data, AdvancedTCA systems impose additional requirements on the OS support, specifically to handle hot swapping defective disk drives (or entire storage boards) and to re-synch the replacement drive or board after insertion, without having to take the system down.

The Linux Multiple Devices (MD) driver along with other user space RAID tools is an example of a conforming software RAID 1 implementation.

Hot Swap Support

In addition to managing redundant devices, the OS on AdvancedTCA systems should also support hot swapping of individual devices and even entire boards. A key requirement on the OS that arises as a result of devices and platforms being hot swapped is the device naming requirement described below.

Persistent Device Naming

A fully utilized AdvancedTCA solution will need to provide seamless removal and insertion of components without adversely effecting the larger telecommunications application. A CGL-conforming OS will provide a policy-based subsystem for naming of transient components. This will allow long-running HA applications to gracefully deal with changing resources without the need to restart applications, or reboot the OS.

Carrier-Grade Linux Support for HA Middleware

The HA middleware discussed earlier relies on a set of services provided by the OS. This section deals with some of those services and the requirements they impose on CGL.

Full Kernel and System Library Support for IPMI 1.5

Having Intelligent Platform Management Interface (IPMI)-enabled hardware is only half the story. Without full support for IPMI in the operating system, solution providers are still forced to depend on costly custom software stacks to create complete HA solutions.

A CGL-conforming OS provides both an IPMI driver exposing a direct connection to the blade's baseboard management controller, along with a user space framework designed for enabling system management software.

In addition to exposing programmatic access to IPMI, a CGL-conforming OS also provides kernel-level primitives for immediately signaling the shelf manager on a critical kernel error via the baseboard management controller over the Intelligent Platform Management Bus (IPMB). This enables the shelf manager to take appropriate action (such as forcing a fast cluster node failover) to maintain service availability targets.

SAF Hardware Platform Interface Support

The SAF is an industry consortium of major players in the telecommunication industry including leading Network Equipment Providers (NEPs), Original Equipment Manufacturers (OEMs), platform and building-block providers, and independent software vendors. The goal of the SAF is to develop a framework and specifications for service availability to enable highly available carrier-grade systems to be developed using off-the-shelf platforms, middleware, and applications. The SAF has published the Hardware Platform Interface (HPI), which is the programming interface between the HA middleware and the platform. Members of the forum are working together to create multiple implementations of the specification. A CGL-conforming OS will provide such a HPI implementation, allowing standardized HA middleware and applications.

The HPI implementation provided with the CGL-conforming OS will provide the next level of complexity on top of the IPMI, utilizing multiple communication paths (IPMB, Ethernet, etc.) as needed to meet the extreme demands of a carrier environment.

Carrier-Grade Linux and TCO

In addition to defining a system that is HA because of the AdvancedTCA focus, CGL requirements also decrease TCO in a similar way to AdvancedTCA by being open and standards-based. CGL reduces the cost of obtaining hardware and software (OS and applications) and reduces maintenance time and unplanned downtime.

Reduction of Hardware and OS Costs

The openness of Linux means that using a Linux distribution with carrier-grade functionality can reduce both hardware and OS costs. Since the Linux kernel supports multiple architectures, customers have a choice of cost-effective architecture options. A CGL distribution is derived from the freely available Linux kernel, and there are multiple CGL distribution vendors competing for market share, so customers have additional freedom in choosing the distribution that meets their cost and feature needs.

Reduction of Application Costs

By detailing the list of features required for a carrier-grade operating system, the OSDL CGL requirements have increased the feature set available in an OS, reducing the need for users to develop and maintain custom solutions in order to implement a full carrier-grade solution. A large number of the CGL requirements are implemented as open source projects, furthering the technology by drawing documentation and bug fixes from the larger open source community. Other requirements are a more natural fit for proprietary implementations based on open interfaces, and are fueling a growing ecosystem of telecom independent software vendors (ISVs). The combination of these two complementing software development models results in a longer lifetime for the technology.

Reduction of Development Costs
CGL requirements, like AdvancedTCA, are also standards-based, which allows for horizontal building-block solutions. The Portable Operating System Interface (POSIX) specification [11] defines a standard interface for an OS; porting from one POSIX-compliant operating system to another should not require costly rewrite, since the same standard interface is used. In addition, other stable Unix[*] interfaces such as Simple Network Management Protocol (SNMP), which is described on the next page, and all the individual interfaces specified in the SAF Application Interface Specification either are currently or are planned to be CGL requirements. In addition to source-level interfaces, the standard binary interface as detailed by the Linux Standard Base (LSB) [12] is also a CGL requirement. LSB's goal is to allow Linux software applications to run on any LSB-compliant system. Customers can therefore choose the LSB-compliant CGL distribution that meets their cost and feature needs and know that Linux software applications can be run on that system.

Reduction of Maintenance Time
In addition to being standards-based, CGL requirements also focus on serviceability, which reduces TCO by ensuring that CGL systems can be maintained, upgraded, and debugged quickly. The SNMP [13] is one such serviceability requirement. SNMP describes a simple, basic, and lightweight network management tool consisting of a Management Information Base (MIB) that contains information about the resources and activity on a node, managers that request these data, and agents that run on the network devices to be managed and that provide MIB data [14]. CGL defines a baseline set of MIBs that provide enough information for a CGL system to be more easily serviced. Another set of CGL requirements that assist in serviceability are those focused on ensuring that sufficient debug information is available after a kernel dump, thus allowing quicker debugging of system failures. CGL requirements focus on enabling the saving of kernel dumps to multiple targets (disks, memory, locations across the network) as well as producing configured dumps with only the data structures the user would like to see. CGL requirements also have been expanded to include the ability to take dump information from the system and applications as well as from the kernel.

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

Reduction of Unplanned Downtime
In addition to maintenance costs, unplanned downtime is also costly. CGL has specific requirements to reduce and prevent unplanned downtime.

Downtime Root-Cause Analysis Features

Regardless of how well a system is tested, unplanned downtime from software or hardware failure is going to happen. An AdvancedTCA-based solution lends itself to uninterrupted service availability via availability clusters instrumented for standard AdvancedTCA features, but the failover hardware is no longer redundant when a failure happens, and costly engineering hours spent tracing the root cause of a failure can quickly eat away at an operation budget.

A CGL-conforming OS provides many mechanisms designed to minimize root-cause analysis downtime by enabling debugging of a live system. Such features include the following:

- Kernel debugger
- Dynamic insertion of debug probes in both kernel and user space executing code
- Standardized kernel messages
- Kernel profiling support
- Kernel summary dump and dump analysis tools
- Live system application dump

Combining these advanced live debugging features that come with any CGL-conforming OS, together with the familiarity of these features to nearly every recent computer science graduate around the world, causes downtime (and therefore downtime expense) to be dramatically reduced.

CGL Security Features

A lack of security is another reason for unplanned downtime, and one CGL requirements section is devoted to security. One basic framework for security that CGL requires is Linux Security Modules (LSMs) [15]. LSM is a Linux kernel framework that supports security modules, primarily access control modules. Customers can create or choose an open source project implementing security modules with the functionality they would like. Access control modules prevent and permit access to objects and processes in a system, and therefore help prevent malicious users from executing programs on a node or from getting access to files on a node.

CONCLUSION

Today’s market requires that carrier-grade solutions be highly available to meet the 99.999% demands on carrier-grade systems as well as have a low total cost of ownership to enable system owners to make a profit. The solution to increasing availability with a cost-effective system requires a hardware, middleware, OS, and application-level solution. AdvancedTCA provides an open, standards-based carrier-grade equipment solution that increases high availability (HA) by enabling a robust failover solution as well as having a robust shelf management definition. The Intel NetStructure Chassis Management Module implements the AdvancedTCA shelf management and can therefore be used to increase the HA of a system. Carrier-grade Linux provides an open, standards-based carrier-grade OS and application solution that increases HA by defining requirements geared at increasing the availability of a system at the OS and application level. Because both AdvancedTCA and Carrier-Grade Linux (CGL) are open and standards-based, they reduce total cost of ownership (TCO) by reducing the dependency on costly vertical, proprietary solutions. A carrier-grade solution involving AdvancedTCA equipment as well as a CGL distribution can help increase HA and decrease TCO.

ACKNOWLEDGMENTS

We thank the following people for their comments, useful suggestions, and support: Chetan Hiremath, Tisson Mathew, Udayan Mukherjee, Mark Skarpness, and Imad Sousou.

REFERENCES

[1].“Go Ahead. High Availability and More: Achieving a Service Availability™ Solution,” http://data.goahead.com/sr/FAQ-fault-management.pdf.

[2].Intel® Corporation, “A Framework for System High Availability,” June 2000, pp. 273389-001.

[3]. http://www.picmg.org

[4]. http://www.advancedtca.org/

[5]. Intel® AdvancedTCA* website: http://developer.intel.com/technology/atca/

[6].IPMI information: http://www.intel.com/design/servers/ipmi/

[7].Service Availability* Forum: http://www.saforum.org/home

[8].HA Middleware: http://www.goahead.com/products/white_papers.htm

[9].CGL requirements: http://www.osdl.org/lab_activities/carrier_grade_linux/documents.html

[10]. Linux kernel: http://www.kernel.org

[11]. POSIX: http://www.opengroup.org/austin/aardvark/finaltext/

[12]. LSB: http://www.linuxbase.org

[13]. SNMP: http://www.ibr.cs.tu-bs.de/projects/snmpv3/

[14]. Stallings, W, “Security Comes to SNMP: The New SNMPv3 Proposed Internet Standards,” *The Internet Protocol Journal,* vol. 1, December 1998, pp. 2, (http://www.cisco.com/warp/public/759/ipj_3.pdf).

[15]. LSM: http://lsm.immunix.org/

[16]. Bonding: http://sourceforge.net/projects/bonding/

[17]. RAID 1: http://en.tldp.org/HOWTO/Software-RAID-HOWTO.html

AUTHORS’ BIOGRAPHIES

Sharad Garg is a modular-server architect at Intel. His research interests include modular storage servers, distributed computing, and distributed file systems. He received M.S. and Ph.D. degrees in Computer Science from the University of Connecticut. His e-mail is sharad.garg at intel.com.

Raj Sistla is a senior software engineer in the Embedded IA Division (ICG/NPG) at Intel. His interests include modular servers and server management. He received his M.S. degree from the State University of New York at Buffalo and is currently working on his M.B.A degree from Indiana University. His e-mail is Raj.Sistla at intel.com.

Ramesh Caushik is a staff software engineer at Intel. His interests include networking, telecom applications, and the Linux operating system. He joined Intel in 1993 and has a M.S. degree in Computer Science from Utah State University. His e-mail is Ramesh.Caushik at intel.com.

Julie Fleischer currently works as a software engineer in Intel’s Telecom Software Program group and participates in the Carrier-Grade Linux Working Group. Her interests include software quality and software process improvement. She received her M.S. degree from Case

Western Reserve University. Her e-mail is Julie.n.Fleischer at intel.com.

Rusty Lynch is a software architect at Intel where he is engaged in various open source activities relating to the Linux operating system in carrier-grade environments. His interest is primarily in core operating system enabling, but he has worked in many other aspects of Linux since joining Intel in 1996. His e-mail is Rusty.Lynch at intel.com.

Addressing TCP/IP Processing Challenges Using the IA and IXP Processors

Dave Minturn, Corporate Technology Group, Intel Corporation
Greg Regnier, Corporate Technology Group, Intel Corporation
Jon Krueger, Technology and Manufacturing Group, Intel Corporation
Ravishankar Iyer, Corporate Technology Group, Intel Corporation
Srihari Makineni, Corporate Technology Group, Intel Corporation

Index words: TCP/IP, network processing, 10 Gbps, acceleration, IXP, IA, IPF, ETA

ABSTRACT

The majority of datacenter applications such as web services, e-commerce, storage, and firewall use Transmission Control Protocol/Internet Protocol (TCP/IP) as the data communication protocol of choice. As such, the performance of these applications is largely dependent upon the efficient processing of TCP/IP packets. In addition, with the arrival of the 10 Gigabit Ethernet, the TCP/IP packet processing needs to become faster and more efficient to let the applications benefit from higher bandwidths. Motivated by this, the material presented in this paper focuses on (a) network bandwidth and the associated TCP/IP processing requirements of typical datacenter applications, (b) challenges of scaling TCP/IP packet processing to 10 Gigabit speeds, and (c) exploring to what extent these challenges can be addressed by the Intel® IA32 and IXP network processors. We also discuss potential techniques that further help TCP/IP processing.

INTRODUCTION

TCP/IP over Ethernet is the most dominant packet processing protocol in datacenters and on the Internet. It is used by both server and network infrastructure applications (e.g., firewall, proxy server, etc.). We show that the TCP/IP component of these applications constitutes a significant amount of the overall processing and hence it is important to understand its characteristics and make sure that it can scale well up to 10 Gbps and beyond. Analysis done on TCP/IP in the past [1] has shown that only a small fraction of the computing cycles are required by the actual TCP/IP processing and that the majority of cycles are spent in dealing with the Operating System (OS), managing the buffers, and passing the data back and forth between the stack and the end-user applications. Many improvements and optimizations have been done to speed up the TCP/IP packet processing over the years. CPU-intensive functions such as checksum calculation and segmentation of large chunks of data into right-sized packets have been offloaded to the Network Interface Card (NIC). Interrupt coalescing by the NIC devices to minimize the number of interrupts sent to the CPU is also reducing the burden on processors. In addition to these NIC features, some OS advancements such as asynchronous IO and pre-posted buffers [12] are also helping to speed up TCP/IP packet processing. While these optimizations, combined with the increases in CPU processing power, are sufficient for processing packets at 100 Mbps and 1 Gbps Ethernet speeds, these are clearly not sufficient for 10 Gbps speeds. Upcoming applications like storage over IP, web services, and ubiquitous broadband connectivity to homes enabling video-on-demand applications require multi-Gbps speeds that require TCP/IP packet processing to scale to these speeds with the least amount of computing power. Figure 1 shows the transmit and receive-side throughput and CPU utilization data for the Microsoft Windows* 2003 Enterprise Edition* TCP/IP stack running on an Intel® Xeon™ processor (2.6 GHz) with

® Intel is a registered 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.

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

Hyper-Threading Technology. These measurements were taken using NTttcp, a variant of the ttcp micro benchmark [11] optimized for the Windows OS.

This paper is organized as follows. We perform an in-depth measurement and evaluation of packet processing on Intel's state-of-the-art server and network processors. The primary focus of this study is to (a) quantify the computational requirements of the TCP/IP packet processing component within server and network infrastructure workloads, (b) identify and analyze the challenges of scaling the TCP/IP packet processing to 10 Gbps in both the server and network infrastructure environments, and (c) show how Intel server and network processors are addressing some of these challenges by discussing the relevant Intel research.

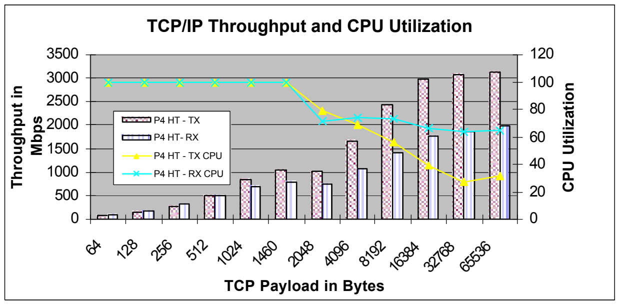


Figure 1: TCP/IP performance on Intel® Xeon™ processor

TCP/IP PROCESSING IN THE DATACENTER

In order to understand the impact of the TCP/IP processing on datacenter applications, we first need to characterize the networking requirements of these applications. In this section, we study applications ranging from firewall applications and load balancers (that primarily run on networking infrastructure equipment) to front-end web servers, application servers, and back-end database servers. We discuss the TCP/IP requirements of these workloads.

Server Workloads

The networking requirements for server workloads can be characterized by experimenting with commercial benchmarks such as SPECweb99 [3], TPC-W [4], and TPC-C [5]. Figure 2 illustrates the types of incoming and outgoing network traffic for these benchmarks.

To represent web server workloads, we chose SPECweb99 since it is a benchmark that mimics the load (70% static and 30% dynamic requests) in a web-server environment. A primary requirement for SPECweb99 is that the network rate needs to be within 320 Kbps and 400 Kbps per connection. Based on measurement and tuning experiments[1] in our labs, we find that the rate on a dual Intel Xeon processor-based system is around 330 Kbps. Based on this estimate, Figure 3 shows the network bandwidth requirements for the web servers as a function of the achieved performance (number of simultaneous connections; x-axis is normalized to 4000 connections).

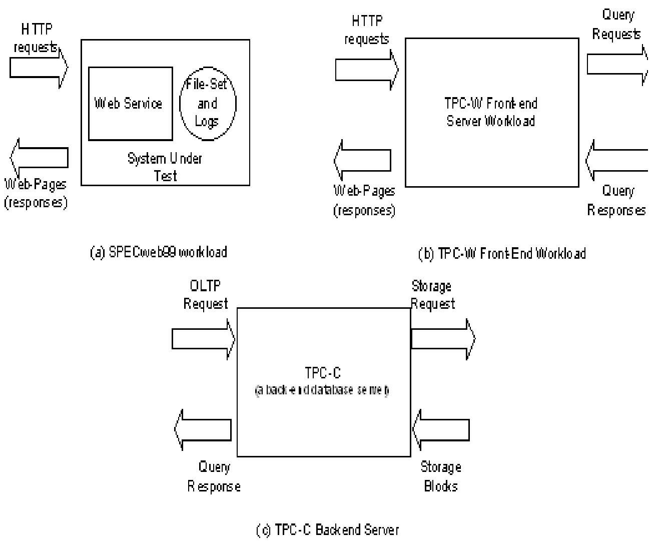


Figure 2: Networking content in server workloads

The relationship of network rate to performance is basically linear. It should also be noted that SPECweb99's network traffic is dominant on the transmit-side, with an average application buffer size (average file size) of roughly 14 Kbytes.

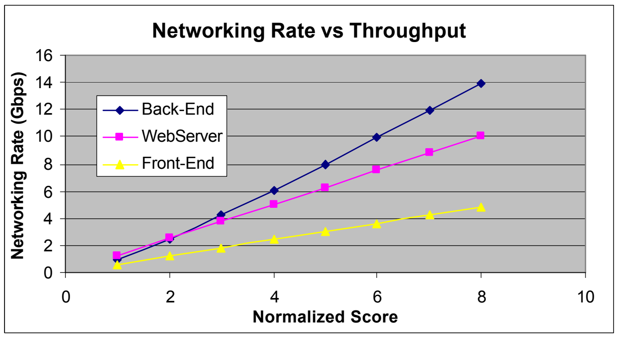


Figure 3: Networking rate for server workloads

To study front-end servers, we analyzed TPC-W, an e-commerce benchmark that models an online book store. TPC-W specifies 14 different types of web interactions that require different amounts of processing on the system(s) under test (SUT). Each interaction requires multiple requests and responses (for the HTML content, the images, and other data) between the clients, the web server, the image server, and the back-end database server. TPC-W experiments[1] in our lab (for a scale factor of 10000 items) show that the SUT receives roughly 2.5 Kbytes of network data from the clients, while it transmits around 46 Kbytes of data to the clients. Based on these experiments, we also find that this outgoing network data is typically broken up into roughly 24 Kbytes per transaction transmitted by the web server and 22 Kbytes per transaction transmitted by the image server. The breakup of incoming network data from the clients is around 1.6 Kbytes to the web server and 0.9 Kbytes to the image server. Based on this data, in Figure 3, we show how the web server network bandwidth changes with an increase in throughput (measured in web interactions per second (WIPS); x-axis is normalized to 3000 WIPS).

Next, we study the networking content in a back-end database server by analyzing TPC-C, an online-transaction processing benchmark that is based on fulfilling database transactions in an order-entry environment. The purpose here is to study the potential networking requirement for storage-intensive servers that access storage blocks over TCP/IP. Based on measurements in our labs, we have estimated the number of IO accesses per transaction as a function of the overall throughput. Figure 3 shows the potential networking requirements of back-end servers as a function of overall performance (specified in number of transactions per minute (tpmc); x-axis is normalized to 40000 tpmc). It should be noted that the storage IOs per transaction depend not only on the performance level but also on the memory capacity available on the platform. It should also be noted that the application buffer size requested from the storage device is basically a physical page (8 Kbytes in our configurations).

Having studied the networking rates and payload sizes for these workloads, we also measured the number of instructions that are required to transmit or receive these payloads over TCP/IP (on a server running Windows Server 2003). Reference [6] describes the details of these experiments. Figure 4 shows the number of instructions (path length) required to transmit or receive various payload (buffer) sizes using TCP/IP. Based on the path length per payload as shown in Figure 4, the number of transmit and receives operations per transaction and the number of instructions per transaction for these three benchmark applications, we estimate the TCP/IP (data path) component. This is illustrated in Figure 5. We find that web server and front-end servers may spend around 25% to 30% of their path length for TCP/IP processing. The packet processing component for back-end servers can be as high as 35% to 40%.

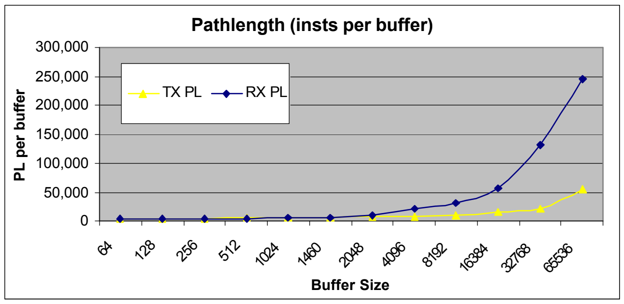


Figure 4: Path length for TCP/IP processing

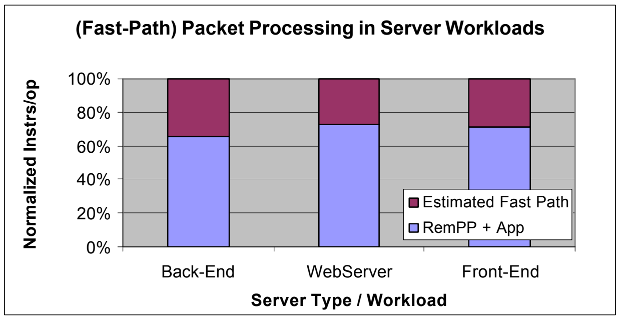


Figure 5: TCP/IP processing in server workloads

Network Infrastructure Workloads

Workloads in the network infrastructure are typically quite different from the workloads processed by end devices such as servers and workstations. Infrastructure equipment has the primary task of moving packets towards their final destinations. Examples of infrastructure equipment include routers, firewalls, and gateways.

As packets get closer to their final destinations in the Internet, the infrastructure equipment can get much smarter in its decision-making process to forward a packet.

Load balancing, URL redirection, L4 switching, web caching, and L7 switching are services that can be found in modern infrastructure equipment, and these services require a much deeper understanding of the TCP header and TCP payload in order for a forwarding decision to be made.

Layer 4 load balancing attempts to equally distribute the packet processing load amongst a group of end node servers. Packet forwarding decisions are based on the Layer 3 IP addresses and the Layer 4 TCP port numbers of arriving packets. The TCP payload does not need to be examined. Similarly, a Layer 4 switch will make its

[1] It should be noted that the commercial benchmark experiments that we discuss do not necessarily meet the strict reporting and auditing rules nor do they fully conform to the specification. Their use here is only to understand behavioral characteristics and should be treated as such.

circuit establishment, resource allocation, and port forwarding decisions based entirely on the IP addresses and TCP port numbers of arriving packets.

URL redirection is similar to Layer 4 load balancing in that the intent of the URL redirection is to equally distribute the web page server load amongst a set of end nodes. But, because each packet must be inspected for the destination URL, the TCP payload must be at least partially reassembled and searched for each packet received.

Layer 7 switches require a deep inspection of the TCP payload of each arriving packet in order to make a forwarding decision. The Layer 7 switch will establish circuits, allocate resources, and forward packets based on the IP addresses, TCP port numbers, and some other field or tag that has been identified in the TCP payload of each arriving packet. An example would be a Layer 7 switch for business e-commerce transactions. The e-commerce transaction switch would search each TCP payload for a specific XML tag or set of XML tags before making the final forwarding decision.

Unlike an end node that processes the entire TCP payload of each arriving packet, infrastructure equipment can process all, some, or none of the TCP payload depending on the types of services being provided.

Infrastructure equipment that processes TCP payloads will eventually forward the packets received. The transmitted packets are similar in size if not identical to the received packet sizes. The end result is that TCP processing in infrastructure equipment is symmetric with respect to receive TCP processing vs. transmit TCP processing. The symmetric TCP processing of infrastructure equipment can be quite different from the TCP processing found in end nodes. In a typical web server application for example, the client requests being received can be quite small, while the web pages being transmitted can be quite large.

TCP/IP PROCESSING IN SERVERS

In this section we discuss the major challenges faced by the servers in scaling the TCP/IP packet processing to multi-Gigabit rates without consuming excessive amounts of processing power. We then show how the Hyper-Threading Technology built into the Intel Pentium® 4 processors help address these challenges to some extent. We also highlight some other potential techniques that help in this regard.

Challenges

TCP/IP processing on servers presents multiple challenges that are specific to their role and environment. A server's main role is to source and sink requests and data to and from clients and to do work on their behalf. This requires that the server terminates one or more connections from potentially many clients, simultaneously. The processing required to efficiently terminate client connections is the main TCP/IP processing challenge for network servers. A basic rule-of-thumb has long been used to estimate the network performance of servers. This rule-of-thumb states that it takes one hertz of CPU frequency to process one bit of TCP/IP network traffic in one second. For example, a 1 GHz processor should be able to source or sink 1 Gbit of TCP/IP network traffic. This "hertz per bit" rule is a generalization and only applies to the best case (bulk transfers) and in many cases the ratio becomes very large as seen in Figure 6 [2]. Even in the best case, it is questionable whether it is acceptable to expend 20 GHz of CPU processing power to service TCP/IP on a full-duplex 10 Gigabit link.

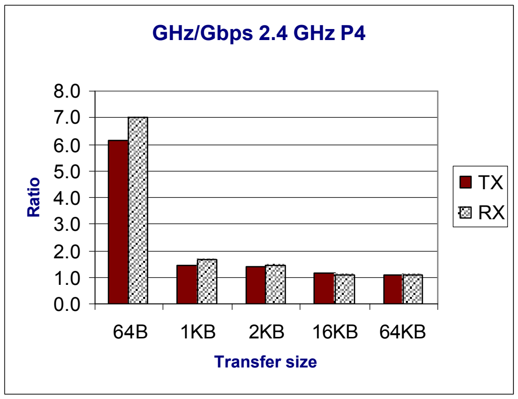


Figure 6: Measured GHz per Gbit

The challenge for the server environment is to enable the scaling of TCP/IP network bandwidth at the lowest possible CPU utilization. Low CPU utilization is important in order to leave sufficient processing cycles for useful work. There are multiple factors that make this a formidable challenge.

One major challenge for servers is the *buffering and copying of the data* carried by TCP/IP. These data copies are largely due to legacy applications and interfaces. The predominant application interface to TCP/IP networks is the socket interface. Traditional Berkeley sockets are synchronous in nature, and tend to favor the spending of

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

CPU cycles for generality. Even though some operating systems and applications have developed more performance oriented, non-blocking, application interfaces, many existing applications still rely on basic sockets because they are common across platforms and operating systems. The result is that some optimizations, in particular avoiding data copies, cannot be implemented while still supporting a large body of user application code. Figure 7 is a high-level view of the end-to-end data path of a TCP/IP data transfer showing how the data at each end is copied by the CPU in order to effect the transfer.

The next major server challenge deals with *operating system integration and overhead*. It has long been shown that operating system overheads are a major factor in the performance of TCP/IP on a server [1]. The TCP/IP software stack on a server needs to share the CPU, memory, and IO resources with the OS. In order to do this, it needs to live by the rules of the OS in terms of scheduling, resource management, and interfaces. OS mechanisms such as system calls, preemptive scheduling, layered drivers, and interrupt processing, as well as data copying, are all sources of overhead that limit the efficiency and performance of TCP/IP on servers.

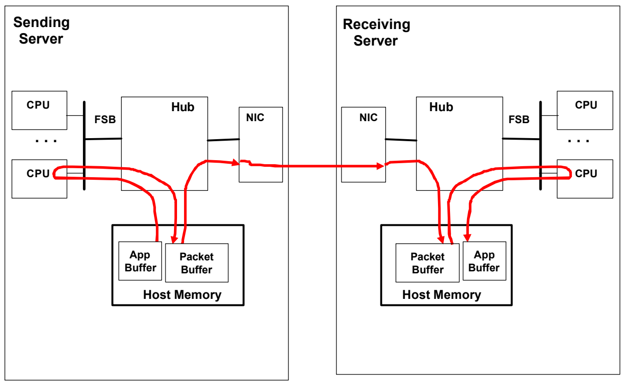


Figure 7: End-to-end data path

Figure 8 shows an example of the processing breakdown for TCP/IP transmit. In this test case, the simple TTCP test program was transmitting 1 Kbyte packets on a dual processor server running the Linux* operating system. It clearly shows that the TCP/IP processing is not the dominant portion of the overall processing.

Another challenge for TCP/IP processing at the server is *memory latency and poor cache locality*. Server memory and cache architecture is optimized for the efficient execution of a broad set of applications. The TCP/IP software stack generally exhibits poor cache locality due largely to the fact that it deals with large amounts of control information and data that are entering or exiting the system through the network interface. For example, an incoming packet is placed into memory by the network interface controller, then writes its status to a descriptor in memory, and generates an interrupt.

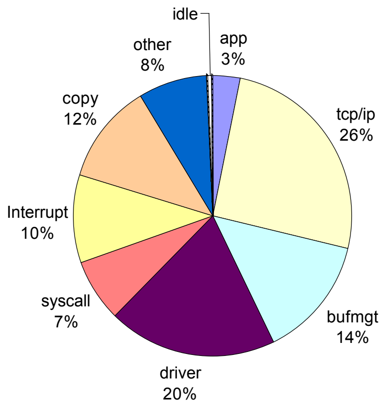


Figure 8: Server processing profile

The TCP/IP software must examine the control information including the descriptor and packet headers for each packet, and since the packet was just placed in memory, this will always cause a cache miss and thus a CPU stall. The data carried by TCP networks also exhibits poor temporal cache locality because new data are continuously being sent or received [14] and it has been shown that IO-intensive workloads cause a large number of memory stalls due to cache misses [15]. Given the rapid advances of network link technologies, combined with the growing disparity between processor speeds and memory latency (see Figure 9), this problem becomes a larger challenge over time.

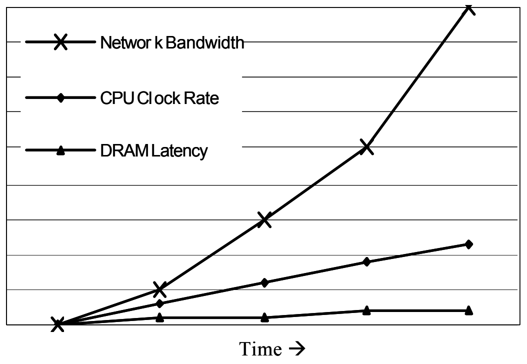


Figure 9: Performance improvement trends

Addressing the Challenges

In order for servers to scale to 10 Gigabits of TCP/IP performance and beyond, the problems of data copies, operating system overheads, and memory latency must be addressed. As in many complex systems, these issues and their solutions are inter-related.

One method to alleviate the need for data copies is to extend the sockets interface to allow non-blocking and/or asynchronous interfaces. The fundamental idea is to be able to post transmit and receive buffers without blocking the application, and to synchronize the application and the networking stack when the operation is complete. This allows the underlying networking stack to prepare the application buffers and move data into or out of them while the application makes forward progress. There are examples where progress has been made in this area. Microsoft has extended the basic socket interface with their Winsock API that includes capabilities such as overlapped IO and completion ports. There are additional ongoing developments to extend the socket semantics such as the Open Group's Extended Sockets interface, and the Linux AIO (Asynchronous IO) extensions. Changing and extending long-standing APIs is a difficult and long-term problem because of the large number of existing legacy applications.

Another method for avoiding data copies is through protocol extensions. An example of this is the ongoing industry-standard effort to define Remote Direct Memory Access (RDMA) [13]. The RDMA proposal adds protocol layers that enable the data to carry information about their ultimate memory location, thus eliminating the need to copy them at the end station. The RDMA model has been implemented in previous networks [8, 9], and this is now being adapted to IP transports such as Stream Control Transmission Protocol (SCTP) and TCP.

Finally, rather than trying to avoid the data copy problem altogether, it may be possible to optimize it such that it is not a limiting factor to application performance. The ability to move data more efficiently via the use of advanced instruction set features, or to be able to overlap the data movement operation with other useful computations are possible means to optimize the data copy operation.

Another major server challenge to be addressed is *operating system overhead and integration*. As seen in Figure 8, the actual TCP/IP processing component is relatively small, while the combination of interrupt processing, system calls, data copies, buffer management, and other OS-related processing make up the bulk of the overhead.

To address the OS overhead issue, there has been an ongoing effort to offload large portions of the TCP/IP networking stack from the OS onto a device. These devices, generally called TCP Offload Engines or TOEs, generally offload the bulk of the TCP and IP processing combined with the Layer 2 media controller. TOE devices have at least two challenges to overcome. First, they are dependent upon interface support from the operating systems that they are trying to offload. Another challenge for TOEs is that they tend to make tradeoffs between cost, performance, and flexibility. TOE devices that are fixed function, or hard coded, promise the best performance, but lack the flexibility afforded by software implementations. Processor-based, programmable TOE devices are more flexible, but have performance limitations, depending on the capabilities of their processing core(s), and they tend to be more expensive because of the processor and associated memory costs.

The Embedded Transport Acceleration (ETA) project [7] at Intel Research and Development has explored the partitioning of multi-processor systems in order to isolate the network-related processing from the operating system. The ETA prototype has shown that an Intel processor that has been partitioned away from the host operating system can perform TCP/IP processing more that twice as efficiently as a normal symmetric multi-processing system. Figure 10 shows the relative amount of CPU work that is expended for several pieces of the TCP/IP processing pie such as TCP/IP itself, buffer management, driver processing, and other OS-related processing.

Yet other server challenges to address are the issues around memory latency and cache locality. There are at least two methods that can be used in order overcome this challenge: concurrent processing and extensions to existing cache coherency protocols.

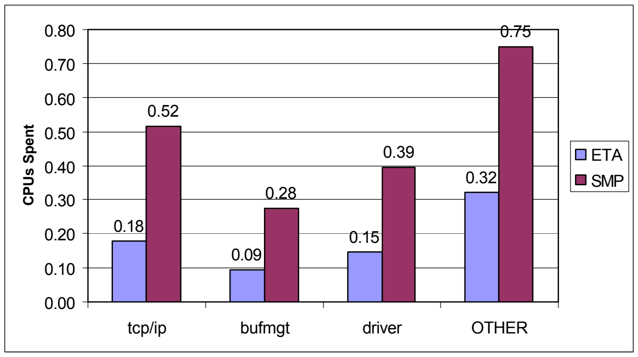


Figure 10: ETA processing efficiency

Server networking workloads tend to allow a great deal of concurrent processing. Most servers must deal with a large number of TCP/IP streams, and a packet of a given stream has minimal dependencies on the packets of another stream, and thus can be processed simultaneously. Concurrent processing of TCP/IP streams can be accomplished using multiple threads. Thread concurrency allows a given processing unit to hide the latency of memory accesses by overlapping the processing of a given packet while the processing of another packet is stalled waiting for memory. Figure 11 is a conceptual diagram of overlapping useful "work" with memory stalls when threads are processing packets from multiple TCP/IP streams.

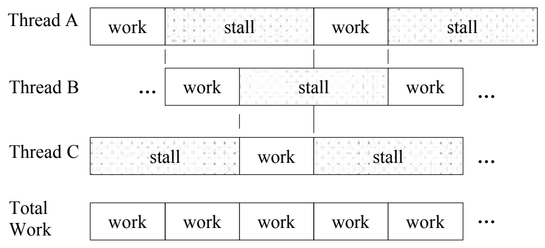


Figure 11: Thread concurrency

The Intel Pentium 4 processor supports Hyper-Threading (HT) Technology, a form of simultaneous multi-threading (SMT) that allows a single processor to appear as two (or more) logical processors. SMT allows multiple instruction streams to be issued simultaneously in order to more efficiently fill the CPU's instruction pipeline in the event of a stall, including stalls caused by long memory access latencies. We measured the effect of the HT multi-threading on simple throughput tests and saw roughly 30% to 50% improvements in throughput for both transmit and receive, as seen in Figure 12.

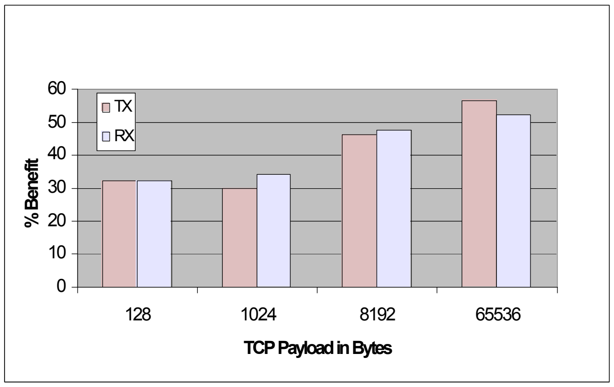


Figure 12: Benefits of SMT multi-threading

Memory stalls during packet processing can also be avoided by extending the semantics of certain memory transactions. If incoming data could be placed directly into the cache of the CPU that needs to do the TCP/IP processing, memory accesses could be avoided that would otherwise cause stalls in the CPU execution pipeline. Examples could include the control information, such as device descriptors and TCP and IP packet headers, as well as the data that are the payload of the packets. We have simulated the effects of directly depositing packets into the cache of the CPU to examine these effects. Figure 13 shows simulation results of one possible scenario where the total time to process a given TCP/IP packet could be reduced substantially if the entire packet was deposited directly into the CPU cache by the network interface. In this case, the memory references to the Network Interface Card (NIC) descriptor and the packet headers (the packet read latencies) would be replaced by much faster cache references, as well as the memory read references used to copy the payload data.

TCP/IP PROCESSING IN NETWORK INFRASTRUCTURE

In this section we focus on the TCP/IP processing in the network infrastructure. We present major challenges that limit the scalability of the TCP/IP packet processing to multi-Gigabit rates and show how these challenges can be overcome to a significant extent with systems based on Intel IXP2850 network processors.

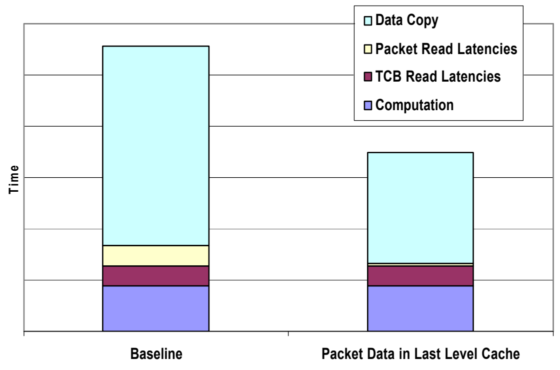


Figure 13: Effect of direct cache references

Challenges

TCP/IP processing performed by infrastructure devices presents an interesting set of challenges. Some of the challenges are a result of the complexity of TCP processing itself; for example, TCP checksumming. But most of the challenges presented in processing TCP relate to the environment in which TCP resides. Examples of these types of challenges include hiding memory latencies, lookup engines that can scale to millions of entries deterministically, and secure platforms with encrypted TCP payloads.

We highlight these challenges by showing an example of a converged firewall platform built using the Intel IXP2850 Service-Specific network processor. The converged firewall performs the normal packet forwarding and filtering that a standard firewall performs with the added features of IPSec for platform security and intrusion detection. The intrusion detection feature requires deep packet inspections into the TCP payloads. The IPSec feature requires us to perform decryption or encryption on every packet.

Our converged firewall can have millions of TCP connections open simultaneously. The resulting memory bandwidth that can result from processing packets for each of these connections can be daunting. This memory bandwidth is a combination of the packet data being moved into and out of memory, and the TCP context for each connection being read, modified, and written back to memory. Also, since our converged firewall is performing intrusion detection services on every packet received, this further increases the bandwidth utilization that the converged firewall will consume.

The chart in Figure 14 depicts typical memory bandwidth requirements while running a simple firewall forwarding application with TCP in a 10 Gigabit Ethernet environment. Note that the chart shows memory bandwidths resulting from simultaneously receiving and transmitting packets at various sizes at 10 Gigabit rates.

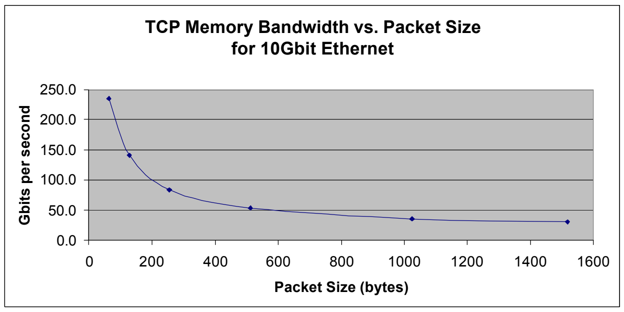


Figure 14: Memory bandwidth requirements

Related to the bandwidth of the memory being moved is the associated latency with waiting for a particular memory transaction to complete. For example, if a firewall thread in the IXP launches a memory read to fetch TCP state associated with a new packet which has arrived, the thread must wait for the associated TCP data to return from the memory controller before it can continue to process the packet. While it waits for the TCP data, the thread can perform no processing on the newly arrived packet. The thread must either process data associated with a different packet, or it must swap out and go to sleep.

Typical memory latencies for various IXP DRAM memory accesses are shown in Figure 15.

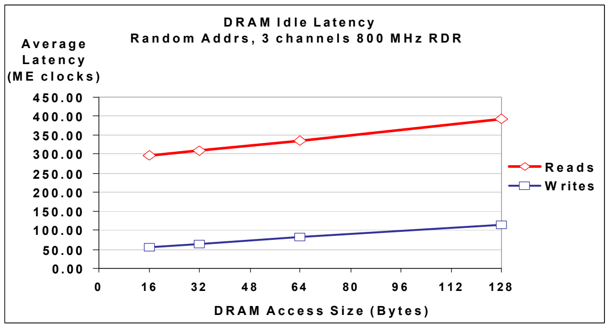


Figure 15: DRAM access latencies

By comparison, the chart shown in Figure 16 illustrates the packet arrival times of various small frame sizes as network speeds increase. Note that at 10 Gigabits per second, the packet arrival times for small frames are considerably smaller than the smallest DRAM read latency shown in Figure 15.

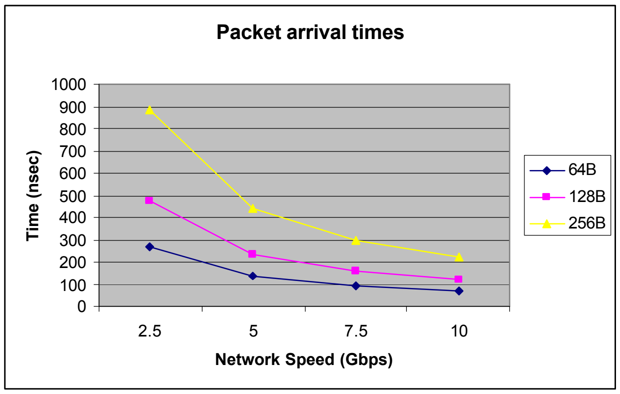


Figure 16: Packet arrival times

Context lookups while processing TCP connections can become problematic as the number of concurrent TCP connections grows. Context lookups are required to search for TCP state associated with a received TCP/IP packet. For a converged firewall that must support one million concurrent TCP connections, the challenge is to construct a lookup engine that will remain deterministic in returning a result with all one million connections established.

One of the difficulties in implementing a TCP state search is that lookup tables can require several dependent DRAM reads in order to resolve a single search. The chart in Figure 17 details the average and worst-case latencies to perform a TCP context search using a simple hash table scheme. Note that the worst-case latencies approach two orders of magnitude difference as compared to the small packet arrival times illustrated in Figure 16.

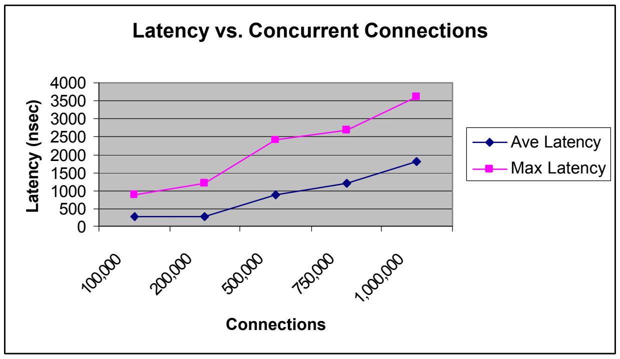


Figure 17: Context lookup latencies

TCP checksum and copying TCP payload data to an application are both problematic for our converged firewall. Both the checksum and the payload copy operate on every byte of the packet, and both will require a number of memory accesses in order to complete. The number of memory accesses is dependent on the size of the packet; the larger the packet the more memory accesses necessary. The memory accesses to complete the TCP checksum and the payload copy will increase both the memory bandwidth utilization and the total latency of our converged firewall.

The IPSec feature of our converged firewall will require us to decrypt a large portion of the arriving packets, and to encrypt a large portion of packets being forwarded. This can be problematic as most crypto routines are extremely computationally expensive, and quite complicated to implement using general-purpose CPUs. Crypto acceleration hardware is available from several third-party vendors. However, the bus interfaces to these chips may not provide the bandwidth or latency to allow us to achieve our 10 Gbps crypto rates, or the buses may be proprietary and require special hardware to interface with.

Addressing the Challenges

A firewall application built using an IXP2850 addresses the challenges of TCP/IP processing in several ways.

The IXP provides multiple memory interfaces that each use an independent set of internal buses to deliver data. The DRAM interface provides three channels of RDIMM access for a total DRAM bandwidth of 38 Gbps for reading or writing. The SRAM interface provides four channels of quad data rate (QDR) SRAM access for a total bandwidth of 48 Gbps for reading and writing. The Scratch interface provides a single channel of standard SRAM access for a total bandwidth of 22 Gbps for reading or writing. The end result is a memory subsystem capable of delivering over 100 Gbps of total memory bandwidth.

By allocating our TCP data structures across the various memory interfaces, we achieve a very good distribution of bandwidth utilization. This bandwidth efficiency yields very high returns as our packet processing rates increase.

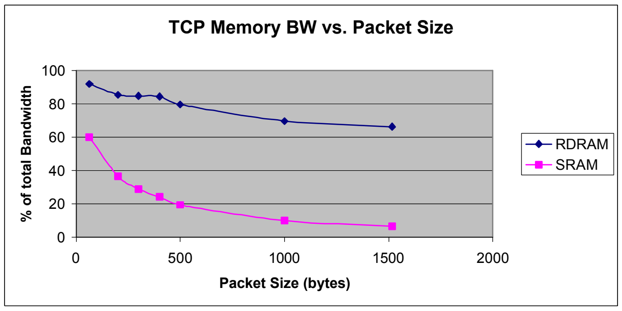


Figure 18: Memory bandwidth headroom

To help address the memory latency problem the IXP provides multiple processing cores called microengines, with each microengine having up to eight threads. The multiple threads allow us to hide the latency associated with the IO transactions of packet processing. Threads that have completed their memory transactions are running and processing while at the same time threads that are waiting for memory transactions to complete are sleeping.

Figure 19 illustrates an example of multiple threads hiding latency and computes associated with packet processing:

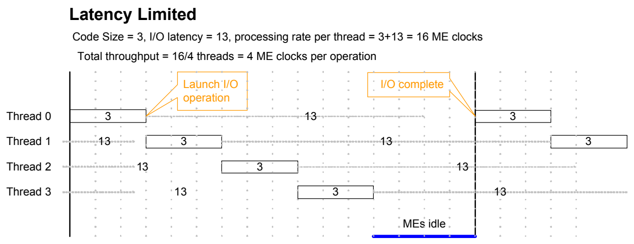

Latency Limited
Code Size = 3, I/O latency = 13, processing rate per thread = 3+13 = 16 ME clocks
Total throughput = 16/4 threads = 4 ME clocks per operation
Launch I/O operation
I/O complete
Thread 0
Thread 1
Thread 2
Thread 3
3
13
MEs idle

Figure 19: Multithreading

The first case to consider is Thread 0 running alone. In this case the total operation requires 16 microengine cycles, and there are 13 idle cycles while waiting for the IO transaction to complete. In the scenario with all four threads running over the same 16 microengine cycle period, we have processed four times more data, and reduced the total idle time to 4 microengine cycles.

The IXP implementation of the firewall application provides several hardware mechanisms useful in helping solve the TCP context lookup problem. Context lookups by the firewall are made more deterministic by implementing connection tables using multiple fixed size arrays, and using well-behaved hash algorithms to distribute the established TCP connections evenly amongst the various fixed size hash buckets.

The IXP provides a hardware hash unit based on an irreducible polynomial. The hardware hash unit is used to return well-distributed hash results that in turn are used to generate offsets into the various hash tables. Also, the irreducible nature of the hash polynomial allows us to store a small portion of the hash result with each hash bucket to greatly simplify lookup hit and miss decisions.

The hash tables are distributed across both SRAM and DRAM to maximize bandwidth efficiency and to take advantage of parallelism in the hash table accesses across multiple threads. Finally, each IXP microengine contains a hardware Cyclic Redundancy Check (CRC) unit capable of reducing very large key sizes to 32-bit keys to streamline the use of the hardware hash engine for both IPv4- and IPv6-based addressing.

The chart in Figure 20 shows the resulting average and worst-case lookup latencies using the advanced search functionality of the IXP firewall. Note the improvement over using the search engine based on a simple hash table design shown in Figure 17.

The IXP provides hardware to assist in accelerating TCP checksum calculations for arriving packets. This greatly simplifies all TCP checksum calculations as the checksum can now be computed incrementally.

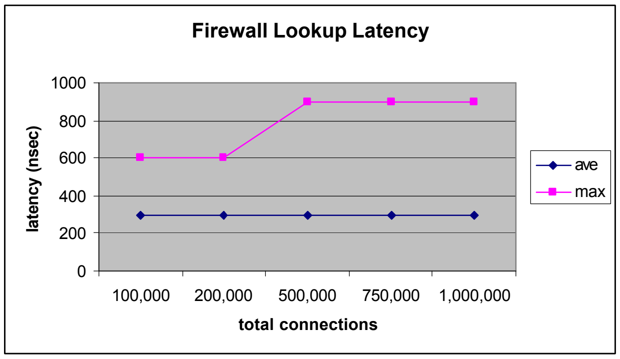


Figure 20: Firewall lookup latencies

The dependency on the payload length is eliminated, and checksums for both incoming and outgoing TCP segments can now be calculated by a function dependent solely on the length of the packet headers, which is a much more deterministic calculation. This greatly reduces both the memory bandwidth and the latency associated with computing the TCP checksum.

Figure 21 compares the read bandwidth required to compute TCP checksums for various frame sizes at 10 Gbps using (1) software only and (2) the hardware acceleration provided by the IXP-based converged firewall.

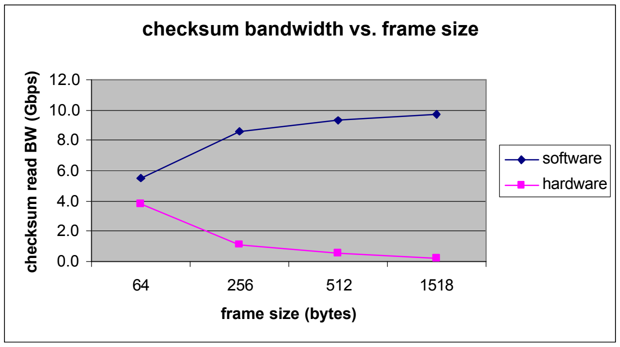


Figure 21: TCP checksum bandwidth

To help in eliminating data copies, the IXP provides hardware mechanisms to register or pre-allocate memory buffers in the system. This allows the packet-processing functions of the firewall to use the same set of allocated memory buffers as the security functions without incurring additional overhead. The firewall can pass all data to and from TCP by reference without the need for performing data copies.

To address the challenges of performing high-speed crypto, the IXP contains an integrated cryptographic unit capable of performing DES, 3DES, and AES crypto calculations. The crypto unit can also perform SHA-1-based message authentication. The integrated crypto allows the converged firewall to achieve 10 Gbps encrypt and decrypt rates without having to access an external bus or have special hardware to interface with the crypto functions.

CONCLUSIONS

Accelerating TCP/IP is essential to allow network-centric applications to benefit from the increase in the Ethernet bandwidth. There are several challenges that currently prevent TCP/IP processing from scaling to these speeds. On servers, the major challenges faced include memory latency, O/S overhead, and data copies. The impact of memory latency can be overcome to a large extent with multithreading techniques (such as Hyper-Threading Technology). The overhead associated with data copies can be minimized by using advanced streaming instructions or specialized hardware for copying data asynchronously. In addition, intelligent prefetching techniques such as placing the incoming data directly into the cache (DCA) also help. For network infrastructure equipment, the major hurdles include memory latency, memory bandwidth issues, and connection look-ups when supporting thousands of simultaneous connections. These are mostly overcome by the highly parallel Intel IXP2850 network processors with a high-bandwidth memory subsystem. Special hardware assists like checksum and hashing engines on these network processors enable TCP/IP processing at 10 Gbps rates.

ACKNOWLEDGMENTS

We express our thanks to Michael Espig, Ramesh Illikkal, Annie Foong, and Ram Huggahali for their contributions to the server networking studies. We also thank Raed Kanjo, James Mead, and Will Auld for providing us insights to the measurement and analysis of commercial workloads. We thank Dale Juenemann, Wajdi Feghali, and Matt Adiletta for their contributions, guidance, and inspiration on TCP/IP performance studies on the IXP.

REFERENCES

[1] D. Clark, et. al., "An Analysis of TCP Processing Overhead," *IEEE Communications,* June 1989.

[2] A. Foong, T. Huff, H. Hum, J. Patwardhan, and G. Regnier, "TCP Performance Analysis Re-visited," *IEEE International Symposium on Performance Analysis of Software and Systems*, March 2003.

[3] "SPECweb99 Design Document," available online at http://www.specbench.org/osg/web99/docs/whitepaper.html

[4] "TPC-W Design Document," available online on the TPC website at www.tpc.org/tpcw/

[5] "TPC-C Design Document," available online on the TPC website at www.tpc.org/tpcc/

[6] S. Makineni and R. Iyer, "Performance Characterization of TCP/IP Processing in Commercial Server Workloads," to appear in the *6th IEEE Workshop on Workload Characterization*, Austin TX, Oct. 2003.

[7] G. Regnier, D. Minturn, G. McAlpine, V. Saletore, and A. Foong, "ETA: Experience with an Intel Xeon Processor as a Packet Processing Engine," *Hot Interconnects 11*, August 2003.

[8] D. Cameron and G. Regnier, *The Virtual Interface Architecture*, Intel Press, 2002.

[9] InfiniBand Trade Association, *http://www.infinibandTA.org*.

[10] *Optimizing Intel IXP2800 RDRAM Performance: Analysis and Recommendations*, Intel Corporation, Version 001, October 2003.

[11] "The TTCP Benchmark," http://ftp.arl.mil/~mike/ttcp.html

[12] A. Jone, J. Ohlund,. *2002 Network Programming for Microsoft Windows Second Edition*, Microsoft Press, Redmond, Washington.

[13] RDMA Consortium, http://www.rdmaconsortium.org/home

[14] B. Ahlgren, M. Bjorkman, and P. Gunningberg, "Towards Predictable ILP Performance – Controlling Communication Buffer Cache Effects," December 1995.

[15] B. Bershad and J. Chen, "The Impact of Operating System Structure on Memory System Performance," *The 14th Symposium on Operating System Principles*, 1993.

AUTHORS' BIOGRAPHIES

Dave Minturn is a senior architect in the Corporate Technology Group at Intel. He has 20 years of experience in the computer industry and has been with Intel since 1989. He has worked on parallel super computers, IO subsystems, networking protocols, and file systems. He is currently investigating various threading models and other optimizations to efficiently scale TCP/IP packet processing. His e-mail is dave.b.minturn at intel.com.

Greg Regnier has been at Intel since 1988 and is currently in Intel's Corporate Technology Group. Greg spent the first part of his Intel career in the Supercomputing Systems Division where he worked on the development of the message passing subsystems of the Touchstone Delta and Paragon parallel supercomputers. Greg was part of the core architecture team that developed the Virtual Interface Architecture. He co-authored a book on the Virtual Interface Architecture with Don Cameron that was published in 2002. His recent work has concentrated on improving the scalability and performance of the datacenter network using standard networking media and protocols. His e-mail is greg.j.regnier at intel.com.

Jon Krueger is a senior architect and software engineer with the Polymer Memory Group at Intel. He has worked in the field of networking for the past 14 years, and has been specializing in applications running on network processors for the past four years. His most recent focus has been in the area of TCP acceleration. Jon holds B.S. degrees in both Electrical Engineering and Computer Science from Oregon State University. His e-mail is jon.krueger at intel.com.

Ravishankar Iyer has been with Intel since 1999 and is currently with the Communications Technology Laboratory. Since joining Intel, he has worked in the areas of Internet server architecture and performance, commercial workload characterization, peer-to-peer computing, and network acceleration. Most recently, his work has been focused on characterizing and enhancing packet-processing performance in servers. Previously, he also worked for the Computer Systems Laboratory at Hewlett Packard Laboratory and in the Server Engineering Group at Intel Corporation. He received his Ph.D. degree in Computer Science from Texas A&M University in 1999. His research interests are in the areas of Internet/networking protocols, computer architecture, distributed computing, and performance evaluation. He has published over 30 papers in these areas. His e-mail is ravishankar.iyer at intel.com.

Srihari Makineni is a senior software engineer in the Corporate Technology Group at Intel. He joined Intel in 1995 and has worked on video conferencing, multimedia streaming, web/e-commerce applications, and system and server management technologies. His areas of interest include networking and communication protocols and computer architecture. His current work is focused on developing techniques for accelerating packet-processing applications on IA-32 and IPF architectures. He holds a Masters degree in Electrical and Computer Engineering. His e-mail is srihari.makineni at intel.com.

Distributed Control Plane Architecture for Network Elements

Manasi Deval, Intel Labs, Intel Corporation
Hormuzd Khosravi, Intel Labs, Intel Corporation
Rajeev Muralidhar, Intel Labs, Intel Corporation
Suhail Ahmed, Intel Labs, Intel Corporation
Sanjay Bakshi, Intel Labs, Intel Corporation
Raj Yavatkar, Intel Communications Group, Intel Corporation

Index words: control plane protocols, network element, scalability, high availability, network processor

ABSTRACT

With the explosion in the use of the Internet for e-business, entertainment, and wireless communication, Internet control and routing infrastructure are facing limitations in terms of the increasing scale. Because the Internet is now mission-critical to many industries, providing high availability (HA) is a major concern.

In this paper, we discuss the scaling limitations of the routing and signaling protocols used in the Internet and propose a novel method to distribute the functionality of control plane protocols implemented in network elements such as switches and IP routers. The proposed method relieves the scalability bottlenecks in various signaling and routing protocols by selectively off-loading certain periodic operations of the protocols to the data plane processing components within a network element. This leads to scalable, highly available, and robust signaling and routing protocol implementations. We use the popular routing protocol, Open Shortest Path First (OSPF) as an example to discuss the appropriate metrics for evaluating the performance of an OSPF implementation, and then we identify the bottlenecks in scaling-up an OSPF implementation. We also describe a generic method for distributing the functionality of routing and signaling protocols, such as OSPF, across both control and data planes that makes them (a) more scalable, (b) more resilient in the presence of faults, and (c) amenable to faster convergence in the face of link or node failures.

INTRODUCTION

Imagine waiting at a 4-way stop sign at a crowded intersection and wondering if a traffic signal would be more efficient. The data and communication traffic on the Internet is directed and controlled by network elements such as IP routers or switches that operate like stop signs and traffic signals. The engineering and design of these network elements is akin to managing traffic at a busy intersection.

As vehicular traffic increases, the roads have to be widened, additional traffic signals have to be added, and the efficiency of signals has to be improved to provide better traffic management. Network traffic, too, is constantly growing because newer services are constantly being deployed to attract more consumers. Newer services lead to an increase in both control and data traffic. Similarly, in order to support the increasing amount of network traffic, the network elements have to support a growing number of physical interfaces leading to a higher density of network elements in any given area (i.e., ISP networks).

Current network elements, such as an IP router, a Layer 3 switch, or a 3G Radio Network Controller, consist of three operational planes: control, forwarding, and management. The control plane executes various signaling, routing, and other control protocols in addition to configuring the forwarding plane. The forwarding plane performs the packet processing operations such as IP forwarding and classification, at or close to the line-rate. Typically, the forwarding plane consists of specialized ASICs or programmable network processors that implement per-packet operations. The control plane, in contrast, typically executes on a centralized, general-purpose processor. The management plane, which provides an administrative interface into the overall system, consists of both software executing on a general-purpose processor as well as probes and counters in the hardware. For example, in an IP router, the control plane executes control protocols such as OSPF [6], RIP [7], RSVP [8], RSVP-TE [17], etc. These control plane protocols exchange updates with their peers across the network and generate control information such as routing tables or label tables, needed by the forwarding plane to actually forward data packets. There may be multiple forwarding planes to handle a large number of interfaces.

A typical network element in the current ISP network has hundreds of physical or virtual interfaces. There may be 300-500 network elements in an ISP network. Such large numbers of interfaces per network element, when coupled with an increase in the volume and variety of control traffic, limit the scalability of the control plane in terms of its ability to quickly handle control plane messages or to react in a timely manner to events such as link failures. These problems are further compounded when a network element occupies a critical position in the network topology where the reliability of such a node is critical to the operation of the rest of the network.

In this paper, we focus on improving the scalability of the routing and signaling protocols in terms of the ability to handle hundreds of interfaces and the associated large volume of control protocol messages and events. For the purposes of this paper, the term *protocol* means a routing or signaling control protocol unless specified otherwise.

We first describe the evolution of network elements and the challenges in designing a control plane. We then discuss some concepts in protocol distribution that are further developed into a Distributed Control Plane (DCP) architecture. The DCP architecture is a distributed software framework that can be used to distribute some of the control plane protocol functionality onto the forwarding planes, thus providing a scalable solution for protocols on the network element. As we will see, this distribution is easily possible and adaptable for forwarding planes that are built using programmable network processors. And finally, we describe a case study of distributing the functionality of the OSPF protocol in the context of the DCP architecture.

EVOLUTION OF NETWORK ELEMENTS

Network element architecture, such as the design of an IP router, has evolved more or less in three generations. In the first generation, control and forwarding planes were executed on a common processing resource. For example, several of Cisco's earlier routers used a common Motorola 68000 for both planes. Until very recently, the network element architecture was in the second generation wherein the control and forwarding planes were explicitly separated so that they could be executed on separate processing resources. But these network elements used proprietary interfaces for separation between control and forwarding planes, and they used ASICs for processing in the forwarding plane. However, with the advent of silicon such as network processors [21] and other specialty hardware, came modular component architecture. In such an architecture, network elements could be built using off-the-shelf components from different vendors, and, therefore, a more formal and standardized definition of the interface between control and forwarding planes was required. The network elements that are adopting this standardized and open interface between control and forwarding planes constitute the third or the current generation of the network element architecture. As shown in Figure 1, this separation is done by using open APIs, e.g., those standardized by the NPF [15] and the IETF's ForCES [16], between the control and forwarding planes. The NPF APIs provide a standardized programming interface between control and forwarding planes. The control protocols in the control plane interact with corresponding components in the forwarding plane using these APIs. For example, IPv4 routing protocols in the control plane manage the entries in the IPv4 forwarding table in the forwarding plane via the NPF IPv4 APIs. The IETF is in the process of defining the ForCES protocol that uses a logical model for representing the forwarding plane entities and will provide a protocol to communicate the control data associated with these entities from control to forwarding plane. The protocol will be independent of the forwarding plane data model.

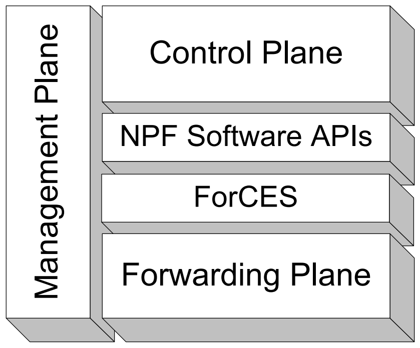


Figure 1: Networking architectural model with separated control and forwarding planes

Figure 2 shows the architecture of a typical second- or third-generation network element. It consists of a control plane (or control card) and multiple forwarding planes (or line cards). All the control plane processing is done in a *centralized* manner by the control card while the entire media-specific processing and line-rate packet processing is done by the line cards.

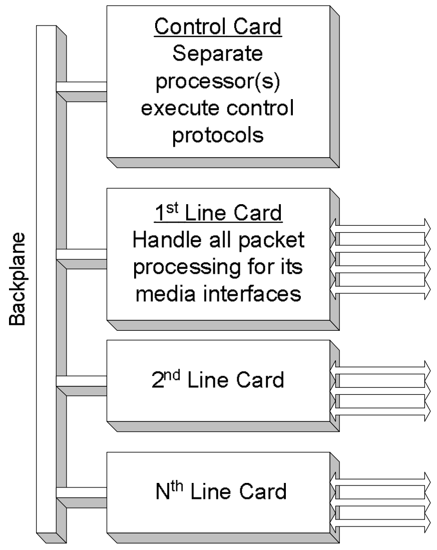

Control Card
Separate processor(s) execute control protocols
1st Line Card
Handle all packet processing for its media interfaces
2nd Line Card
Nth Line Card
Backplane

Figure 2: Current architecture of network elements

Typically a backplane or an interconnect based on ATM, Ethernet, CSIX, or some proprietary switching fabric, connects all the blades and is used to transfer both control and data traffic between the blades. Such an architecture has two key advantages, namely, the processing resources of control and forwarding planes are separated, allowing each to scale independently, and components from different vendors can be used to build each plane because a standardized inter-plane interface is used.

CONTROL PLANE DESIGN CHALLENGES

As the popularity of the Internet has grown, the link bandwidths have grown from megabit rates to gigabit rates, and new protocols have been introduced to support new services being deployed on the Internet.

In response to the growing end-to-end traffic, a typical ISP network now has as many as 300 to 500 network elements (IP routers). Depending upon its location in the network and the services it needs to support, each network element typically runs a wide variety of protocols such as Border Gateway Protocol Version 4 [9], Intermediate System-to-Intermediate System [10], Open Shortest Path First [6], Routing Information Protocol [7], Internet Group Management Protocol [11], Distance Vector Multicast Routing Protocol [12], and RSVP [8]. For example, a typical Digital Subscriber Line Access Multiplexer (DSLAM) runs a complete IP stack and an ATM stack, where each stack is made up of a number of protocols. Moreover, the network elements such as IP routers and Layer 3 switches may contain hundreds of physical or virtual interfaces. As a result, an IP router may have to exchange control information with hundreds of peers. Such a growth in bandwidth, network traffic, and network element density impacts various aspects of the operation of a protocol.

Scalability

As the number of interfaces in a network element increases, the protocol generates more control traffic because it has to communicate with an increasing number of peers via many different interfaces. As newer services are introduced in the network, they either add new types of control traffic to the existing control protocols or introduce new control protocols. Either way it leads to increased complexity and traffic volume in the control plane. In the past, control traffic only accounted for about 2% of traffic in the packet networks, and about 5-15% in the telecom networks. However, with the increase in the number of interfaces and user services supported, this control traffic in packet networks is growing to 5-10% of the total traffic and is expected to be even larger in the telecom networks. For example, consider OSPF [6], a key routing protocol used in most of the ISP networks today. The OSPF protocol periodically exchanges HELLO messages with its directly attached peers. As the number of interfaces or peers increases, the protocol must send, receive, and process a much larger number of such messages. In addition, the OSPF protocol has recently been extended to *OSPF-TE* [19] that supports traffic engineering of the entire network. Such extensions add to the type and number of control messages handled by an implementation of the OSPF protocol.

Contributing to this growth in volume is the per-user state maintained at some network elements. Some protocols, like RNC in the 3G cellular network, establish and manage connections for individual mobile users. Typically, an RNC handles 300,000 to 400,000 users with a projected growth to 500,000 users. Also, with the growing use of VPNs and MPLS, the level of granularity at which forwarding is done in IP routers is moving closer to per-client network level.

All the factors described above contribute to a substantial increase in demand for control plane processing cycles or processors.

Highly Available Protocols

With the growing use of real-time services such as Internet Telephony and video distribution, loss of connectivity for even a second may result in a considerable loss of information and affect the user experience. So network faults need to be isolated and repaired rapidly.

As there are numerous protocols such as OSPF and BGP executing on the control plane, a link event such as link going up or down, typically triggers processing on all or most of them, causing a huge increase in processing requirements, which may saturate the control plane processor and render a network element non-operational. For example, a link going down in an IP router running BGP [9] would mean that BGP would need to update its internal protocol state machine and send appropriate messages to its peers. In addition, it would need to update thousands of routes on its forwarding plane. For a network element to be highly available, not only does it have to react to faults quickly, but it also has to scale up its processing capacity to handle the extra processing needed in the case of faults.

Robustness to Control Plane Denial of Service Attacks

Malicious applications or viruses may launch a Denial of Service (DoS) attack on network elements, so the protocols running on them must be robust enough to operate in spite of such attacks. Many protocols use an authentication mechanism to ensure that the packets received are from valid nodes. An example of a typical DoS attack on a control plane is flooding the control plane with spoofed BGP [9] control packets. BGP packets must typically be authenticated before they are processed. Traditionally, the forwarding plane sends all control packets to the control plane for authentication and processing. If there is a significant increase in the spoofed traffic, the control plane would be flooded with authentication requests causing interference with other control plane processing.

PROTOCOL DISTRIBUTION: CONCEPTS

All of the problems described in the previous section can be alleviated if the control plane processing could be performed in a distributed fashion, such as taking some of the functions and simply distributing them to individual line cards. Such a distribution can deal better with Denial of Service attacks, because processing is now closer to the point of attack and can therefore react quicker and sooner.

We need to bring together the following three elements to achieve distribution of the functionality of a control protocol:

- First, evaluate the internals of a control protocol to identify frequently performed functions that will benefit from offloading to separate processing resources.
- Second, utilize a multi-processor network element hardware platform that provides several options in terms of where a protocol function can be executed.
- Third, define a software architecture and associated infrastructure that enables the creation of a distributed implementation of any protocol.

Evaluating a Control Protocol for Distribution

There are two main issues to be considered while intelligently distributing control processing functions: determining the function(s) of the protocol that can actually be distributed without affecting the correctness of a protocol, and deciding which parts of a network element can host such distributed functions.

What to Distribute? Anatomy of a Protocol

Operations of any control protocol like BGP, OSPF, or RSVP, etc. can be split into three main categories: (1) link-specific functions involving packet forwarding and maintenance of connectivity with protocol peers, (2) protocol-specific processing, such as maintaining the protocol state machine, and (3) functions that update the forwarding plane with the control information.

1. Functions that fall into the category of link-specific functions or independent functions involve parts of a protocol responsible for communication with its protocol peer(s) to maintain the adjacency and status of the connectivity, and parts of a protocol that involve processing of packets on a per-interface basis (for instance, filtering incoming protocol updates from specific protocol peers). These functions are ideal candidates for distribution to processors on line cards. Such distribution helps reduce the computation load on the control processor(s) and bandwidth load on the interconnect fabric, and it also accelerates peer discovery and failure detection.
2. Functions that fall into the category of protocol processing functions are mainly related to computation, based on updates received from the protocol peers to generate new peer updates and new control information, such as new IP routes for installation in the forwarding plane. The protocol state machine maintenance functions belong to this category. These functions need to be considered on a case-by-case basis for distribution, with the focus mainly on reducing the computation load on the control processor(s) when network failures cause additional message traffic.
3. The forwarding plane update or centralized functions consist of updates to refresh the control information in the forwarding plane–for example, route table updates. These functions can rarely be distributed.

Where to Distribute? Types of Protocol Distribution

There are mainly two ways of distributing control plane functionality:

1. *Functional Distribution*. A control plane protocol consists of many distinct components that implement different protocol functions. In the case where a distinct component is independent of other components, it can be cleanly separated from the rest of the protocol functions and be implemented elsewhere. For example, the HELLO messages in the OSPF protocol perform the task of keeping track of the availability of peer nodes or the liveness of adjacent links. Such a function is an attractive target for distribution as it can be cleanly separated from the rest of the OSPF protocol functionality and tends to require frequent communication and processing.

   There are two drivers for functional distribution, (1) the need to separate functions of a protocol that compete for a significant portion of the control CPU capacity, and (2) the desire to exploit additional benefits expected in executing a particular function at a certain location within a system. For example, functions involving physical interface state should be executed on a line card so that the particular function can scale well as the number of physical interfaces increases. The primary goals of functional distribution are to enhance the scalability of a control protocol and make it more responsive to external events by executing certain functions closer to their domain of activity or influence. This type of distribution requires multiple processors in a system. In this paper, we focus on using this distribution method for implementing scalable and robust control planes.

2. *Layered Distribution*. Every network protocol stack consists of multiple layers wherein each layer is responsible for a distinct function and processing of both incoming and outgoing packets. For example, a Node B Application Part (NBAP) layer [20] in a Radio Network Controller (RNC) stack relies on lower protocol layers such as *Iub Frame Protocol*, followed by the UDP/IP layers, and finally, the PPP layer. Because these layers are distinct, it is sometimes useful to perform functions related to one or more of these layers in a separate location to take advantage of the available processing resources. (Even though this is an alternate but valuable way of distributing the control plane functionality, we have chosen to focus on the functional distribution in this paper; we hope, however, to address this method in our future work.)

HARDWARE PLATFORM ARCHITECTURE FOR DISTRIBUTED NETWORK ELEMENTS

Based on the discussion in the previous section, following are the high-level requirements for the hardware platform needed to implement functional distribution within control protocols:

1. It should support multiple processors where control functions can be executed. Typically these processors are either present on line cards and optionally on cards hosting management or other applications.

2. It should be possible to seamlessly scale the compute cycles available for control processing. Typically this is done by adding new line cards or application cards to the network element whenever needed.

3. It should be possible to distribute control protocol functions to an appropriate processor within the system.

As explained in the previous section, there are three classes of protocol functions: Class 1 is link specific (possibly communication or IO intensive), Class 2 is protocol specific, processing or compute intensive, and Class 3 is centralized. Thus, in order to implement functional distribution for such control functions, a hardware platform with three processing levels or tiers would be needed. The lowest or first processing level (for example, processors on line cards) could execute Class 1 and optionally Class 2 control functions. The second processing level (for example, processors on application cards) could execute Class 2 and optionally Class 1 control functions. The third tier (for example, the traditional control card) could execute Class 3 and optionally Class 1/Class 2 control functions. The middle or second tier is the optional processing level and may not exist in all systems. It may also not be required if other processing systems are deemed sufficient. This implies that the hardware platform must at least consist of the first and the third processing tiers. Correspondingly, Intel has introduced a multi-tiered hardware platform of network elements as shown in Figure 3 and Figure 4.

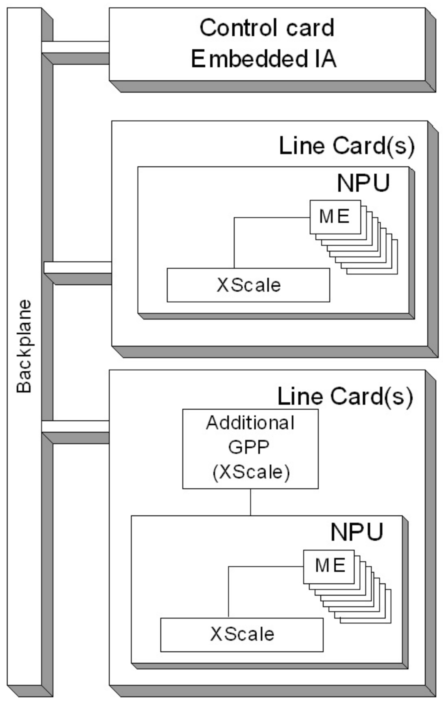


Figure 3: Two-tier network element hardware platform

Figure 3 shows a two-tier hardware platform consisting of a control card built around an embedded IA processor, targeted for compute-intensive tasks (such as execution of control stacks, database lookups, etc.) and multiple line cards built around IXP network processor(s) targeted for communication-intensive tasks (such as wire-speed packet routing, forwarding, etc.). The IXP network processor constitutes multiple microengines and one core processor based on Intel XScale® technology. For example, the IXP2400 has eight microengines [13]. There are two variants of the line cards existing today, namely, line cards with one or more IXPs, and line cards with an additional standalone general-purpose processor (GPP) (e.g., the Intel XScale). Besides the processor on the control card in such a hardware platform, control functions can be distributed to and executed on one or more of the IXPs or on a standalone GPP core, as the case may be. Thus, such hardware platforms provide additional compute cycles for control plane processing whenever a new line card is added to the network element, which is a basic premise for constructing scalable network elements. As the control processing functions can be distributed across processors present in line cards, the computation bottlenecks that exist in the centralized control card are either minimized or removed. In addition to this, the system's response time to external events is reduced by distributing functions closer to wire on line cards. For example, line cards built using the IXP2800 [21] have support for accelerating compute-intensive cryptographic processing. This feature could be used to delegate cryptographic functions within a control protocol to the IXP2800. Lastly, it is quite possible that some network elements may have two control cards for high-availability purposes wherein one control card is active and the other acts as a standby. We are not proposing to use the standby card for offloading purposes.

In order to augment the two-tiered hardware platform further, Intel has also introduced a new family of control plane processors [14]. These processors can be used to add another tier to the two-tiered hardware platform as shown in Figure 4.

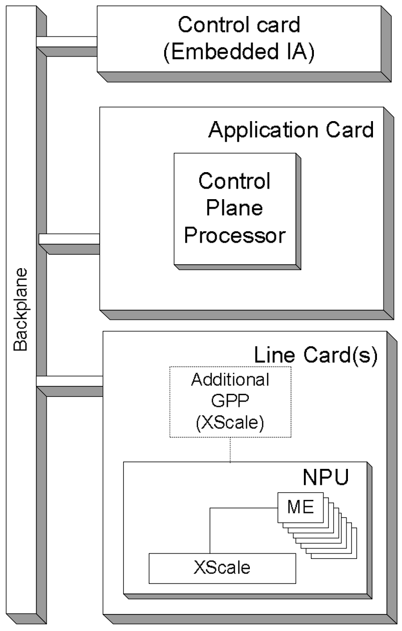


Figure 4: Three-tier network element hardware platform

The control plane processors are targeted for certain compute-intensive communication problems (e.g., deep packet inspection, encryption, etc.) and for compute-only problems (e.g., executing certain control stacks). Control plane processors have features such as low power, low heat dissipation, and smaller die size that make them ideal for the implementation of application cards. Another possible configuration for this hardware platform is to have the control plane processor co-exist with the IXP network processor(s) on the line cards instead of some other GPP.

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

DISTRIBUTED CONTROL PLANE (DCP) ARCHITECTURE

In this section we describe the Distributed Control Plane (Software) Architecture and see how it supports functional distribution of protocol implementations. Some of the key design considerations for this architecture are described below:

1. *Protocol independence*. The architecture should provide the primitives that are common across all kinds of control protocols.
2. *Pluggable architecture*. The architecture should provide well-defined interfaces so that a new protocol implementation can be cleanly plugged in. The protocols use these interfaces to discover processing target capabilities and exchange related protocol-specific information between the components of an implementation.
3. *Multiple processor hierarchies.* The architecture should be able to scale to multiple processor resources either on different processing targets, on the same processing target, or on the same processor core, as in the case of multi-core CPUs.
4. *Interconnect independence*. The architecture should provide an interconnect-agnostic interface to communication protocols. Depending upon the system configuration, two control protocol functions may communicate over different types of interconnects.

A distributed network element has one or more control cards and multiple forwarding plane blades. The DCP architecture consists of two major components: the *DCP Infrastructure Module* and the *DCP Communication Library*. A Control Point (C-Pt) is defined as a control or forwarding plane executing a DCP Infrastructure Module (DIM) along with one or more routing or signaling protocol functions. The DIM maintains the current view of the network element, which entails keeping track of the capabilities and resources of other C-Pts in the network element and the Control Plane Protocol Modules (CPPMs) running on each of them. A protocol implementation that executes a single function or the complete functionality of a protocol as per an RFC or an industry standard is known as a CPPM. Multiple CPPMs work in conjunction to provide a complete implementation of the protocol. The protocol function that has been separated out of a core protocol implementation forms the *Worker CPPM* while the core functionality of the protocol is known as the *Controller CPPM*.

The Distributed Communication Library (DCL) is linked with the CPPM and is responsible for transparently providing communication between the CPPMs. In our implementation, the DIM and CPPM run in different processes. The DCL communicates with the DIM to discover the peer CPPMs, and it provides an abstraction of the peer CPPMs. The common interface between peer CPPMs is known as the CPPM Peer Interface.

Figure 5 shows the operation of a distributed protocol on a DCP-enabled Network Element. Each C-Pt has a DIM and a CPPM running on it. The control points and the CPPMs are connected over the same interconnect.

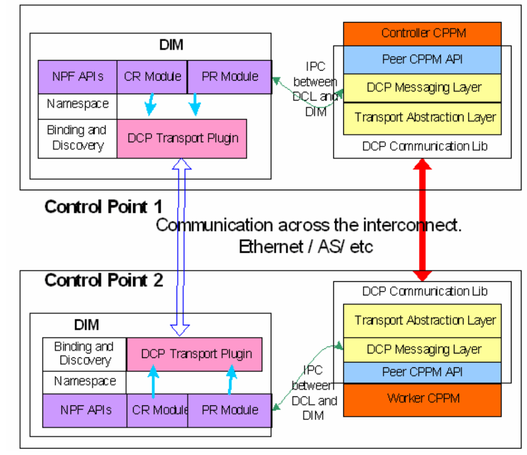


Figure 5: Operation of distributed protocol on a DCP-enabled network element

DCP Infrastructure Module

The DIM contains the logic required by a C-Pt to discover other C-Pts present in a system, establish connectivity with them, and exchange information about the C-Pt capabilities (such as a list of CPPMs registered and their capabilities). The DIM is configured with a policy to allow a C-Pt to communicate with selected C-Pts.

As seen in Figure 5, the DIM is made up of the namespace, Binding and Discovery module, CPPM Registration (CR) module, and the Packet Redirection (PR) module. It also provides a standardized interface to protocol stacks via the NPF [15] APIs. The namespace maintains a logical view of the network element and its components. Various modules register with the namespace to receive appropriate events when there are changes to the namespace. The Binding and Discovery module discovers other C-Pts, exchanges configuration information with them, and then updates the namespace with this information. The information exchanged includes the properties of control points, and the capabilities of the CPPMs running on them.

The CPPM Registration (CR) module allows a CPPM to register its capabilities and get information about its peers. A CPPM uses the Packet Redirection (PR) module to specify the protocol packets of interest to itself. If no such filters are set, the Controller CPPM receives all protocol packets by default.

The DCP Transport Plug-in is responsible for communication between the DIMs. This layer hides the specifics of the physical interconnect and associated transport protocol that are used to exchange information between two DIMs. It provides a connection-oriented interface to set up connections and transfer data between any two DIMs in the system and takes care of managing underlying transport protocols and interconnects. New plug-ins can be easily added to support additional interconnects and associated transport protocols.

DCP Communication Library

The DCP Communication Library (DCL) provides an abstraction of the peer CPPMs in the form of the resources they own or control. It is primarily responsible for transparent communication between the CPPMs and for implementing advanced policies for load-balancing and fast-failover. At initialization, the DCL is configured with CPPM capabilities, such as CPPM, CPPM function, resources, and the interconnect. The implementation of the DCL is independent of the CPPM function or protocol that it is linked to. The operation of this library is akin to an intelligent RPC mechanism.

The DCP messaging layer controls the actual communication between the CPPMs across the Peer CPPM API. It provides functions to serialize and de-serialize data, and transparently exchange it between the associated CPPMs over the network. This layer uses information from the CRM to set up and manage the communication channels among the associated CPPMs. This layer can detect CPPM and connection failures and is informed about the changes in the configuration of peer CPPMs via events from the DIM. Depending on the policy configured at initialization, DCL implements load balancing, redundancy, and dynamic failover. The DCP messaging layer uses the Transport Abstraction layer to set up, manage connections, and provide interconnect/transport independent communication.

The DCL thus hides the details of the interconnect, the transport layer, and the configuration of various CPPMs to allow for easy integration of existing protocol stacks. It also provides support to a CPPM for load-balancing or fast-failover, which is fundamental to a highly available CPPM implementation.

The Distributed Control Plane Architecture is, thus, a generic architecture that enables transparent and seamless distribution of control protocol functions across multiple processor levels in the system. The architecture enables distribution of functionality away from a hotspot closer to places in the system where it can function best, making the system highly scalable. The architecture is independent of the interconnect technology or the processor hierarchy in the system and provides well-defined interfaces so that different control protocols can be plugged in cleanly and easily.

CASE STUDY: THE OSPF ROUTING PROTOCOL

In order to validate our distributed architecture, we chose the Open Shortest Path First (OSPF) protocol, one of the most widely deployed intra-domain routing protocols in the Internet. OSPF is a link state routing protocol, which means that it maintains a database describing the entire network topology of the domain, and it uses this database to calculate the shortest paths to all the nodes in that network. Functionally, the OSPF protocol consists of two major parts: (1) creating adjacencies between neighboring routers in order to exchange routing information and, (2) exchanging this information and synchronizing the databases between the routers. The Hello protocol in OSPF is responsible for establishing and maintaining neighbor relationships. OSPF uses Link State Advertisements (LSAs) to exchange the database information between routers. After synchronization of the database, OSPF uses Dijkstra's Shortest Path First (SPF) algorithm to find the shortest path to all routers/nodes in the network.

In order to monitor liveness of peer routers, each OSPF router exchanges Hello packets once every 10 seconds and each OSPF router considers a neighbor dead after it fails to respond to three consecutive Hello packets. Thus the adjacency failure detection time for OSPF-based routers currently deployed in IP networks is about 40 seconds. This means that any faults in the network or changes in the topology will be detected by the OSPF routers only after 40 seconds. In today's gigabit networks, this would mean the loss of a large amount of data traffic until another path is restored. This slow convergence has motivated discussion around millisecond convergence times and SONET-like restoration schemes for intra-domain routing protocols like the OSPF protocol in standardization forums such as the IETF. Faster failure detection, and consequently, smaller Hello intervals are key to achieving millisecond convergence [1].

We conducted some experiments to find the computation requirements of OSPF running on the Control Plane CPU. We observed the behavior of the protocol under different network sizes and with a reduced Hello interval time. Figure 6 shows our experimental setup. We used an Intel® Pentium® III 1 GHz processor running Linux and the Moy* OSPF protocol code [18] as the device under test (DUT). We connected this Linux router to a system running our simulation network. Our simulator simulated a network of OSPF routers connected in a grid topology. For the purposes of our experiments, we had all the routers including the DUT belong to a single area.

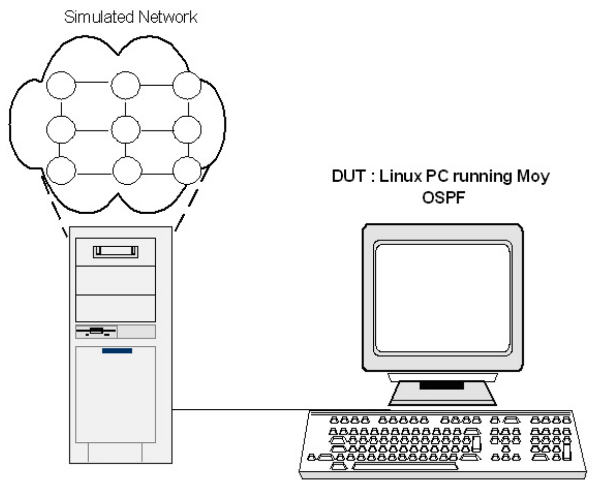


Figure 6: Experimental setup

Our first experiment was aimed at finding how the SPF calculation time varies as the size of the network of OSPF routers increases. Today's operational ISP networks have 300-500 nodes in a single area. For example, as reported in [1], in the year 2001, one of the key AT&T networks had 292 IP routers and 765 physical links connecting the IP routers in a single OSPF area. In our experiment, we varied the number of nodes in the simulation network from between 0-900. A plot of the SPF calculation time on the DUT versus the number of routers in the network for both grid and a mesh network is shown in Figure 7. As expected, the SPF calculation time increases with the network size, since the complexity of Dijkstra's algorithm is *O(elogn)*, where *e* is the number of edges in the network and *n* is the number of nodes. For a grid network topology, *e ~ 2n or O(n);* whereas for a full mesh network topology, *e ~ n*n/2 or O(n*n).* Thus for a full mesh network, the SPF behavior would be more quadratic in nature than for a grid network. Similar results for OSPF's SPF complexity in today's operational networks have been reported by others as well [1], [5].

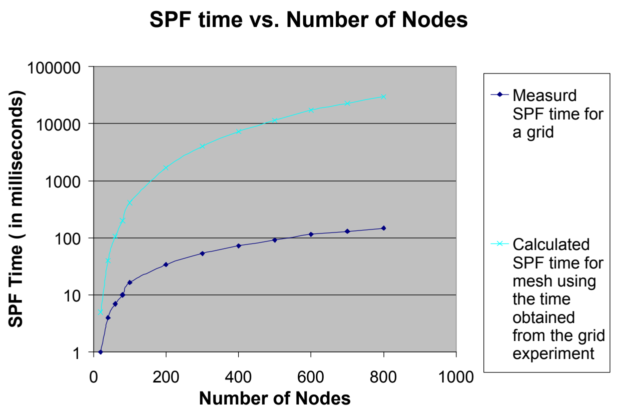


Figure 7: SPF calculation time

Our second experiment measured the CPU utilization of the control plane processor as the *Hello interval* time is reduced. We used 400 nodes in our simulation network to simulate a typical OSPF area size in an ISP. The results are shown in Figure 8. We observed the CPU behavior with only the OSPF protocol, first with the normal *Hello interval* of 10 seconds and corresponding Router *Dead Interval* of 40 seconds (graph 1 in Figure 8), and then with the *Hello interval* reduced to one second and the dead interval reduced to four seconds (graph 2 in Figure 8). For the latter case, we also triggered faults in the simulation network every 1-4 seconds over a period of three minutes to simulate dynamic updates happening in a real network over a short period of time. Potentially, different kinds of faults include events such as *a router going down* and *a link going up or down*. By manually injecting faults, we are just generalizing the effect of any such failure in the network and investigating the behavior of OSPF. Additionally, we wanted to see the impact of back-to-back faults on OSPF. These are rare but when they do occur, they have a significant impact as they either render the router non-operational or require various timers to be introduced in OSPF implementation, making it more compute-intensive.

As can be seen from graph 2 in Figure 8, the CPU load has numerous peaks over a short period of time with the maximum peak at 40%. Note that this is the load on the

®Intel Pentium III is a registered 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.

CPU with only the OSPF protocol code running, in a simple (grid topology) 400-node network. In a real router, there would be other applications running for monitoring, configuration, and management of the router. For Area Border Routers in the Internet, there would be multiple instances of OSPF, one for each, in addition to other protocols running on the Control Plane CPU.

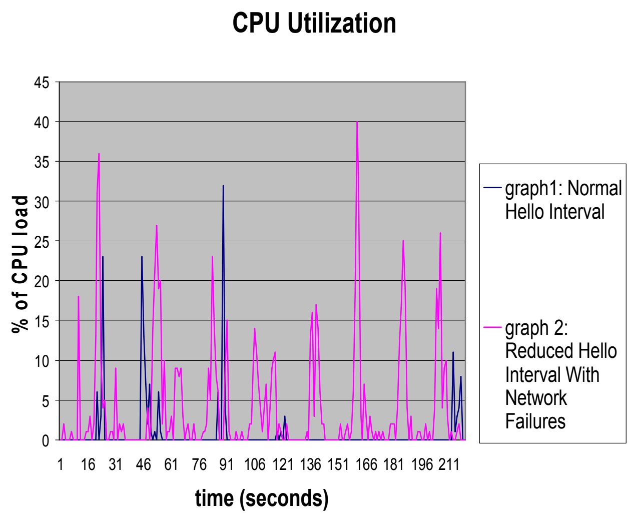


Figure 8: CPU utilization of control plane processor with (a) a normal Hello interval and (b) a reduced Hello interval and network failures

It is clear from these experiments that the demand on the control plane processor by OSPF increases quadratically with network size and a reduction in the *Hello interval* time. Thus, moving to millisecond convergence will further increase the demand on the control plane processor. In order to support millisecond convergence for OSPF routers running in large networks, the OSPF implementation needs to be able to scale with the network size. This is very important because the increase in the CPU load can lead to Hello misses in the OSPF implementation, which could in turn lead to what are called *false adjacency breakdowns* in the network. Such conditions can easily multiply in a congested network and have been known to render an entire routing domain in-operational.

Distributed OSPF

As mentioned before, OSPF functionality can be divided into the Hello protocol and the database synchronization mechanism. For a typical network element consisting of 10 line cards or forwarding elements (FEs) with 10 interfaces each, the CP would need to process 2*3*100 = 600 Hello packets every second (assuming a 333 millisecond *Hello interval* and Hello packets being sent and received at this interval on 100 interfaces). Assuming each Hello packet requires 1 ms time to process, receive, and send a reply, this would mean 600 ms are spent just processing Hello packets on the CP. If we offload this functionality onto each of the FEs, then each FE would need to process only 2*3*10 = 60 Hello packets every second. Thus, in addition to reducing the processing and IO load on the Control Plane CPU, this functional distribution also scales with the system resources (FEs) and does not put all the load on a single FE.

We used the Distributed Control Plane framework to distribute OSPF such that the Hello protocol processing is offloaded on to the forwarding elements or line cards in the network element.

In our implementation, the line cards were composed of two IXP 2400 network processors, and the Hello protocol processing for each line card was offloaded onto the XScale processor. We used an Intel Pentium III 1 GHz processor running Linux as the control plane. The test-bed consisted of a distributed OSPF implementation in a network of non-distributed OSPF routers. The distributed OSPF consisted of a worker CPPM and a controller CPPM. The worker CPPM implemented the Hello protocol, the Designated (and Backup) Router election, and the Interface Finite State Machine (IFSM). All the other parts of the OSPF protocol including the Neighbor Finite State machine (NFSM), maintenance of the Link State Database (LSDB), routing algorithms, etc. are implemented by the controller CPPM. Each CPPM defined a *Peer Interface* via which they exchange configuration, state synchronization, and other messages.

The following messages were sent by the worker CPPM to the controller CPPM using the CPPM Peer Interface:

1. Neighbor parameters of a new neighbor or the changed parameters of an existing neighbor (such as neighbor ID, its idea of Designated/Backup Designated router, etc.).
2. Interface states as a result of the IFSM being executed.
3. Events derived from the received Hello packets such as *HelloReceived*, *AdjOK*, *2-WayReceived* [6] etc. that trigger the NFSM.

Similarly the following messages were sent by the controller CPPM to the worker CPPM:

1. Configuration information such as router ID, interface identifiers on which the Hellos are to be sent and received, type of interface, Hello interval, Hello dead interval, area associated with the interface, etc.

2. Start/Stop Hello on an interface
3. Reset router
4. Neighbor states as a result of the NFSM being executed.
5. *NeighborChange* [6] event that triggers the IFSM.

The above messages are exchanged using DCL primarily in the beginning till the Neighbor is considered *fully operational* [6], after which there are no message exchanges. The worker handles the adjacency maintenance and informs the controller only when critical changes occur in the neighbor parameters. These changes are detected by examining the received Hello packets.

FUTURE WORK

We believe that our prototype version of a distributed OSPF router is a key step towards achieving millisecond convergence in OSPF networks. In order to achieve this, we are working towards offloading the Hello processing onto the IXP's microengines, thereby exploiting the power and programmability of the IXP network processor. By performing the Hello processing in the microengines, we can detect failures faster and run SPF calculations as frequently as required without affecting the load on the control plane processor. We will perform more experiments related to the processor utilization and other performance criteria to further validate our conclusions. In addition, we are also working towards functional distribution of other key protocols such as RSVP-TE and BGP.

CONCLUSION

Increase in network traffic, deployment of new services, and an increase in the number of network interfaces per router are combining to create huge processing and bandwidth demands for control plane processing. In addition, high-availability requirements translate into the need for millisecond convergence when links or nodes fail. Such a convergence requires much more frequent communication among control plane entities. We need novel approaches to scale up the implementation of control protocols to accommodate these needs. We propose a distributed control plane architecture that allows a control plane protocol to seamlessly and transparently move its CPU- or IO-intensive functions away from the control card to other available processing resources within the network element. Our approach offers the following advantages:

1. Faster response to network events such as failure of a link or peer to achieve quicker network convergence. This enables high availability and enhances the robustness of the networks, enabling support for applications that expect deterministic network behavior.
2. Scalability and improved control plane processing speed because CPU-intensive functions, such as encryption and deep-packet inspection, can be offloaded and scaled across auxiliary processing elements in the system. This also results in faster response from the protocol.
3. Resilience towards Denial of Service attacks targeted at the control plane by filtering malicious protocol packets causing these attacks on the line cards or the forwarding plane. This improves the overall robustness of the network element.

Our proposal is based on an experimental analysis based on a simulation study that examined the amount of CPU capacity needed for OSPF computations and handling OSPF messages and events.

Our analysis of the OSPF protocol can also be extended to other routing and signaling protocols. For example, BGP and LMP analysis will likely show authentication bottlenecks whereas an analysis of RSVP-TE would show the need for distributing the state related to each channel once it is established. Similarly, the DCP architecture could be used to offload the encryption/decryption functions of control protocols. We intend to study these aspects further.

ACKNOWLEDGMENTS

We thank Gary Hemminger and Haseeb Budhani from IP Infusion for helping us with data about typical ISPs. We also thank Wayne Allen of Intel Labs for his invaluable help with our experiments.

REFERENCES

[1].Basu, A and Riecke, J., "Stability Issues in OSPF Routing," *ACM SIGCOMM 2001*, UC San Diego, CA.

[2].Alaettinoglu, C., Jacobson, V., Yu, H., "Towards Millisecond IGP Convergence," *NANOG 20*, October 2000.

[3].McPherson, D., Brendel, E. et. al., "Panel: Core Network Design and Vendor Prophecies," *NANOG 25*, June 2002.

[4].Alaettinoglu, C. and Casner, S., "ISIS Routing on the Qwest Backbone: a Recipe for Subsecond ISIS Convergence," NANOG 24, February 2002.

[5].Sheikh, A. and Greenberg, A., "Experience in Black-box OSPF Measurement," in *Proceedings. ACM SIGCOMM Internet Measurement Workshop (IMW),* 2001, pp. 113-125, Nov. 2001.

[6].Moy, J., "OSPF Version 2," *IETF RFC 2328*, April 1998.

[7].Malkin, G., "RIP Version 2 Carrying Additional Information," *IETF RFC 1388*, January 1993.

[8].Braden, R. et. al., "Resource ReSerVation Protocol (RSVP) - Version 1 Functional Specification," *IETF RFC 2205*, September 1997.

[9].Rekhter, Y., "A Border Gateway Protocol 4 (BGP-4)," *IETF RFC 1771*, March 1995.

[10]. Callon, R., "Use of OSI IS-IS for Routing in TCP/IP and Dual Environments," *IETF RFC 1195*, December 1990.

[11]. Fenner, W., "Internet Group Management Protocol, Version 2," *IETF RFC 2236*, November 1997.

[12]. Waitzman, D., Partridge, C., and Deering, S., "Distance vector multicast routing protocol," *IETF RFC 1075*, November 1988.

[13]. http://developer.intel.com/design/network/products/npfamily/ixp2400.htm

[14]. http://developer.intel.com/design/network/products/cpp/ixc1100.htm

[15]. http://www.npforum.org

[16]. http://www.ietf.org/html.charters/forces-charter.html

[17]. "RSVP-TE: Extensions to RSVP for LSP Tunnels," *IETF RFC 3209*, December 2001.

[18]. Moy, J.T., "OSPF Complete Implementation," September 2000.

[19]. "OSPF TE: Traffic Engineering Extensions to OSPF Version 2," *IETF RFC 3630*, September 2003.

[20]. http://www.3gpp.org

[21]. Matthew Adiletta et. al., "The Next Generation of Intel IXP Network Processors," *Intel Technology Journal, Volume 6, Issue 03*, August 15, 2002, pp. 6-18.

AUTHORS' BIOGRAPHIES

Manasi Deval works on defining and developing technology for modular networking and communication platforms at Intel Labs. She received her M.S. degree in Electrical Engineering from the University Of Southern California and a B.E. degree in Electronics and Telecommunication from Maharashtra Institute of Technology, India. Her e-mail is manasi.deval at intel.com.

Hormuzd Khosravi is a senior network software engineer in Intel Labs. He has been involved in defining and developing standards and technology for modular networking and communications platforms at Intel. Previously, Hormuzd held the position of research assistant at C&C Research Labs, NEC USA. Hormuzd received his M.S. degree in Computer Engineering from Rutgers University and his B.E. degree from Bombay University, India. His e-mail is hormuzd.m.khosravi at intel.com.

Rajeev Muralidhar is a senior network software engineer at Intel Labs where he is involved in defining and developing standards and technology for modular networking and communications platforms. Rajeev is also the principal editor of the Packet Handler Task Group at the Network Processing Forum. He received his M.S. degree from Rutgers University and his B.E. degree from the National Institute of Technology, India, both in Computer Engineering. His e-mail is rajeev.d.muralidhar at intel.com.

Suhail Ahmed works as a senior network software engineer in Intel Labs where he is involved in defining and developing technology for modular networking and communications platforms. He received his M.S. degree in Electrical Engineering from SIU, Carbondale and his B.E. degree in Electronics from Bangalore University, India. His e-mail is suhail.ahmed at intel.com.

Sanjay Bakshi is currently an engineering manager and architect in the Intel Corporate Technology Group. He has lead a number of projects related to the usage of the Intel Internet Exchange Processor in various fields such as IP routing, MPLS, 3G wireless, and next-generation control plane architecture. Sanjay received his B.E. degree in Computer Science from the Regional Engineering College, Tiruchirapalli, India. His e-mail is sanjay.bakshi at intel.com.

Raj Yavatkar is currently the chief software architect for the Internet Exchange Architecture at the Intel Communications Group. Raj received his Ph.D. degree in Computer Science from Purdue University in 1989. He is a co-author of the book *Inside the Internet's Resource reSerVation Protocol (RSVP)* published by John Wiley in 1998. He currently serves on the editorial board of the IEEE Network magazine. His e-mail is raj.yavatkar at intel.com.

Advanced Software Framework, Tools, and Languages for the IXP Family

Stephen D. Goglin, Corporate Technology Group, Intel Corporation
Donald Hooper, Intel Communications Group, Intel Corporation
Alok Kumar, Intel Communications Group, Intel Corporation
Raj Yavatkar, Intel Communications Group, Intel Corporation

ABSTRACT

The IXP network processor [1] architecture is designed to process packets at high rates where inter-arrival times between packets are sometimes less than the individual memory access latencies. To keep up with such high arrival rates, IXP presents a novel architecture consisting of a collection of multithreaded, multiprocessors with an explicit memory hierarchy. Such an architecture poses interesting challenges when an application developer wishes to develop software for a wide range of services that can easily be ported from one IXP processor to another. To make it easy to develop portable, modular software on IXPs, we currently provide a combination of development tools and a software framework consisting of libraries, run-time infrastructure, and APIs to facilitate the creation of application building blocks. This paper describes the future directions in IXP programming tools, software frameworks, languages, and compilers.

We first identify the limitations of the current IXA software framework [3], describe extensions to remove such limitations and discuss how these extensions also help in the development process.

Secondly, we provide an overview of the Developer Workbench tool suite and its features, which provide significant advancements over traditional development environments. We highlight features that enable architectural task modeling, allow debugging using bandwidth analysis, thread history, and operand tracing, and provide a novel packet-centric analysis based on extensible protocol packet generation, packet status, and a graphical view of the packet dataflow.

And finally, to move the developers beyond a chip-centric view of programming, we also investigate domain-specific languages and adaptable run-time environments. We describe and justify one such research project under development, called *Baker*, to develop a domain-specific language for network processing.

INTRODUCTION

In this paper, we discuss three new ways that the software development infrastructure for programming network processors is being improved: the existing IXA software framework is being extended, new features are being added to the Developer Workbench, and research in developing a domain-specific language and compiler for packet processing is being pursued. We start with a brief introduction to each topic.

Framework Extensions

Firstly, network processors are being targeted at a wide range of applications with varying packet-processing and throughput requirements. The programmability and flexibility of a network processor make it suitable for applications ranging from Voice over IP to content-based routing with data rates spanning OC-3 to OC-192. In such an environment, the investment made in software development by equipment manufacturers is increasingly significant. Preserving this investment and leveraging it across multiple projects are key considerations when choosing a network processor. The IXA software framework [3] provides the necessary software infrastructure to help develop modular, reusable software building blocks for network processors of the IXP family.

The IXA Portability Framework is now widely adopted for software development on the IXP family. However, the framework is unable to fully support some classes of applications. These applications require support for specific features such as multicast forwarding, handling packet fragmentation, and running at low rates (OC-3 to OC-12). The IXA framework, therefore, must be extended so that it can support all applications efficiently and easily.

The main shortcomings of the existing software framework are (i) lack of support for the creation of multiple packets from a packet in the packet processing stage, (ii) lack of support for efficient manipulation of packet data buffers when they are sent to multiple interfaces, and (iii) the inability of microblocks that were built independently to fit together when run on different (not the same) threads within a microengine.

Development Tools

Secondly, with each new generation of IXP network processors, the hardware architecture becomes more complex: there are additional microengines, memory controllers, and hardware acceleration features. Therefore, the need for advanced tools that simplify the design and debugging processes is greater today than ever before.

In this paper, we describe the current method of debugging IXP applications, then introduce tool enhancements that enable a new, packet-centric, development methodology. We discuss new architectural planning, packet list, history event capture, conditional breakpoint, and instruction operand tracing features.

Next-Generation Compilers

Finally, we are conducting research on a domain-specific programming language, called *Baker*, for developing applications using network processors. Because of the unique hardware features and structure of the network processor platform, conventional, procedural languages are forced to expose many of the details of the hardware to the developer. Exposing low-level details may allow a developer to produce high-performance object code but the development is complex requiring unique skills, and porting an application (or its parts) from one IXP to another is difficult. The Baker language exposes the domain-specific features of packet processing rather than the hardware-specific features. As a result, the code is easier to write and port, and, at the same time, the compiler has sufficient information to be able to produce efficient binaries.

We assume that the reader is familiar with the Intel Network Processor Family [1], previous versions (SDK 3.5 or earlier) of the IXA Software Development Toolkit [2] and the IXA portability framework [3].

FRAMEWORK EXTENSIONS

There are two problems in the existing IXA software framework [3]:

1. The IXA software framework provides the software infrastructure to help develop modular, reusable software building blocks for IXPs. It assumes the model in which a packet traverses from one microblock to another, where each microblock [3] can modify the packet content or the state associated with the packet. In this model, it is not possible to generate multiple packets from a single packet in a microblock. The generation of multiple packets from a single packet in a microblock is required for the support of packet multicast and IP packet fragmentation.
2. The IXA software framework provides the flexibility to port an application on IXPs ranging from the high-end IXPs running at very high data rates to the low-end IXPs running at lower data rates, without any changes to the microblocks. The low-end IXPs have a fewer number of microengines compared to the high-end IXPs. Therefore, several microblocks that may run on separate microengines on a high-end IXP may have to be combined to run on a single microengine when ported to a low-end IXP. However, if these microblocks use microengine *Content Addressable Memory* (CAMs) [1] for achieving efficient mutual exclusion and synchronization, then combining microblocks while they run on different threads in a single microengine is not possible.

We now describe the extensions designed to solve the above two problems.

Support for Multicast and Packet Fragmentation Features

Problem Definition

In the current framework, packet data are stored in DRAM, and every packet has a one-to-one relationship with the meta-data associated with it that is stored in SRAM. Information about a packet is passed across microblocks using a fixed meta-data structure. This framework does not apply in a case where a multicast packet is processed requiring multiple copies of an incoming packet to be sent over different outgoing links. The existing framework requires generating a new packet buffer for each multicast copy. Copying of packet data for multicast is undesirable because it adds to memory bandwidth and latency requirements. Similarly, to handle IP fragmentation, an incoming packet must be divided into parts that are individually sent out as separate packets. The current framework infrastructure would again require copying of data buffers, something that is equally undesirable. Our goal is to extend the framework infrastructure to support new functionality without requiring additional copying of data buffers.

To be precise, we must meet the following requirements to achieve efficient support for multicast and packet fragmentation features:

- A packet data buffer should not have to be copied. Only the packet headers, which are different for all the copies, should be copied when a packet is multicast.
- Only minimal changes should be made to the packet-processing microblocks. Packet-processing microblocks should not have to distinguish among unicast, multicast, or fragmented packets in terms of buffer and meta-data management except when a microblock needs to process the multicast or fragmented packets differently from the unicast packets.
- There should be no performance impact for unicast or non-fragmented traffic. Support for multicast and fragments should not impact the performance of unicast or other traffic requiring no fragmentation.

Extensions to the Infrastructure

When a multicast packet is to be sent out with multiple copies, the only change in each copy of the packet is to the modified header; the payload is left unchanged. The same thing is true when an incoming packet is fragmented and each piece is sent out as a separate packet. Therefore, we extend the framework by adding the concept of *child* and *parent* buffers with the explicit goal of avoiding buffer copies, but still being able to transmit a distinct copy of each packet. Each multicast packet data buffer is a parent buffer. All packets generated from the original packet have a child buffer, containing a new packet header and a pointer to the original parent data buffer.

Zero copy of packet data, except packet headers that are distinct for each packet copy, can be achieved by having the parent buffer that has data buffer and the child buffers, storing only the packet headers, for every copy of the multicast/fragmented packet. The child buffers point to the parent buffer. The child buffers store modified headers for every multicast/fragmented packet. The parent buffer stores the common data buffer for all copies of the multicast packet. The parent buffer contains a *refcnt* (reference count) representing the number of child buffers pointing to it. Figure 1 shows the structure of a multicast/fragmented packet.

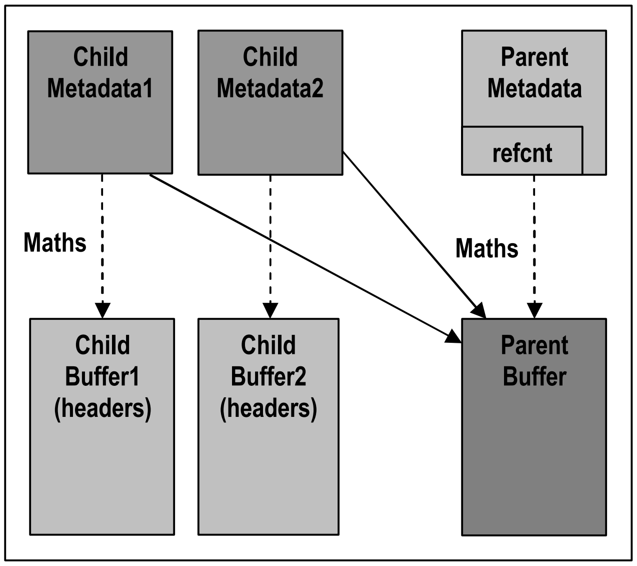


Figure 1: Data layout for a multicast/fragmented packet

Every buffer has a corresponding meta-data. In the current framework, every unicast packet has a corresponding meta-data. We propose that every copy of the multicast packet also has a corresponding *child meta-data*. After a packet is multicast, the *child meta-data* is used by all packet processing blocks and is used for queuing of the packet. This extension needs to be implemented in the architecture in such a way that a child buffer for a multicast packet would not look any different than a parent buffer for a unicast packet. This is achieved by making the child meta-data the same as the parent meta-data except for the fields that are not used by the packet processing blocks. Since most of the packet processing blocks work only on packet headers, they can use a child buffer in the same way as they use a packet buffer for a unicast packet. Thus, there are no changes to the previously written packet processing blocks: multicast or fragmented packets are treated the same as unicast, non-fragmented packets.

As a result of these changes, the *Packet Transmit* block of the packet processing pipeline must change. Before freeing up a buffer, the block first reads the *refcnt* from the parent meta-data, decrements it, and frees the parent buffer only if the *refcnt* is zero. The reading of the *refcnt* adds one extra dependency to the packet transmit code. However, since all packets are stored in the local memory and then processed later, the reading and checking of the *refcnt* can easily be overlapped with the existing memory accesses, after storing a packet in the local memory. This ensures minimal changes to the packet transmit block and makes it possible to meet the line rate in all the cases.

CAM Sharing Extension

Problem Definition

Mutual exclusion is a well-known problem for multithreaded software. For example, in a network processor, several threads may be accessing the same memory location at the same time and updating the value at that location. These accesses and modification to the same location have to be mutually excluded to maintain the correct semantics. Therefore, a network processor needs to have a mechanism to support *mutual exclusion* across these threads. *Folding* provides a mechanism to support this in the IXP2xxx family of network processors. *Folding* uses the microengine CAM to provide mutual exclusion on a location across threads running in a microengine without throttling the performance. Folding is briefly described in the next few paragraphs.

A *critical section* is a part of software that involves a sequence of *read – modify – write* operations on a memory location. The *read* operation reads the value of the memory location into the microengine. The *modify* operation is performed after the *read* operation is complete and modifies the data read from the memory location. The *write* operation writes back the modified data to the memory location. Any other thread can perform the *read* operation on the same location only when the *write* operation of the previous thread is complete. Due to the non-parallel nature of "critical section" code, it becomes a challenge to hide the latencies involved in reading and writing the critical data from/to the memory. *Folding* solves this problem by making sure that the critical data are in the microengine when the second thread needs them. The operations *read – modify – write* are modified in *folding* to allow parallelism. Threads perform all these operations in strict order. For example, thread 0 performs the *read* operation and sends a signal to thread 1 to perform the *read* operation and so on. Thread 0 starts the *modify* operation after all the threads have completed the *read* operation. Similarly, *modify* and *write* operations are performed in strict order.

During *read* operation, a thread does a CAM lookup to determine if the data from the desired memory location are already in the microengine. In the case of a *CAM miss* (data are not in the microengine), the thread issues a read for the data from the memory and updates a free CAM entry to reflect that the data will be available in the microengine. In the case of a *CAM hit* (data are already in the microengine), it does not issue the read from the memory. When every thread is done with the *read* operation, the first thread modifies the corresponding data in the microengine and stores the modified data in the microengine. It then signals the next thread to start the *modify* phase. This ensures that the next threads use the data that have already been modified by the first thread. Hence, only one thread modifies the data at one time and not more than one thread works on the same data at any one time. This ensures mutual exclusion on the data across all threads. A thread performs *write* after it is done with *modify,* only if it is the last thread working on the data. From the above description we see that even for a *critical section*, *folding* allows *reads* and *writes* to be performed in parallel, while allowing mutual exclusion.

Least Recently Used (LRU) is a standard cache replacement policy. In traditional caches that use LRU, when a new entry is to be fetched in the cache, then the entry in the cache that has not been used for the longest period of time is expunged. This scheme of *replacing the cache entry that has not been used for the longest period* is called LRU. In the state of the art, we use *folding* in conjunction with the LRU method to solve the mutual exclusion problem. When we have a CAM miss, we find a free CAM entry using the LRU policy implemented by the CAM hardware to store the new data that are going to be read. We use LRU as a replacement policy in folding because LRU is supported by the CAM hardware and the LRU policy meets the cache replacement requirement of *folding* as long as only one microblock is using *folding* at any time within the entire microengine (across all threads).

However, with the addition of more functionality, especially on medium to low-end performance NPs, more than one microblock runs on different threads in a microengine. These microblocks run independently on different threads and are not synchronized across each other. More than one of these microblocks may need to use the *folding* technique and, therefore, simultaneously use the CAM for the purpose. We show how a single LRU policy across the entire CAM supported by the hardware is not sufficient when the CAM is shared by two or more microblocks running independently in parallel. The single LRU policy across all the CAM entries has the following two problems.

1. *LRU might return a CAM entry that is in use by another microblock.* Suppose, there are two microblocks A and B running independently on a single microengine. Assume microblock A is running on the first four threads and microblock B is running on the last four threads with no synchronization across each other. When needed, the hardware LRU would give the single LRU entry among all the CAM entries in the microengine, even though that entry might be in use by a different microblock running on another thread. We cannot

protect the two different CAM entries, one used by microblock A and the other by microblock B. For example, microblock A may execute its *read – modify – write sequence* for an arbitrarily longer period compared to the one executed by microblock B. Therefore, a thread running microblock A may need to keep a CAM entry, *e1*, much longer than the CAM entries used by the threads running microblock B. Therefore, while entry *e1* is used by a thread running microblock A, threads running microblock B can run much faster and use every other entry, making all of them more recently used compared to entry *e1*. This makes *e1* the LRU even though it is still in use. As a result, the hardware victimizes the entry *e1* and makes it available to a thread running microblock B even though the original thread is still using *e1*. This leads to unpredictable and undesirable results.

2. *A thread running a microblock does not know how to evict an entry that has the data of other microblocks.* Suppose, we have two microblocks A and B running independently and microblock B gets a free entry that has some data previously used by microblock A. Microblock B has to flush the data stored in that entry before it can use the entry for its own data. Microblock B has no idea about the data structures of microblock A and their actual location in the memory. Therefore, it is impossible for microblock B to flush the old data.

In addition to the problem of using a single hardware LRU when two independent microblocks share the CAM in a microengine, the microblocks also need their keys to be unique to ensure that any key of microblock A does not match a key of microblock B. In the following section, we propose a solution to the two problems we have outlined.

Extensions to Support CAM Sharing

A solution to the problem of ensuring uniqueness of keys across microblocks is to assign keys so that a key of a microblock does not match a CAM entry that stores the same key of another microblock. Our proposal is to append a unique microblock ID to each key so that keys from different microblocks can never be the same. We propose that the unique ID be a four-bit constant assigned to each microblock by the dispatch loop [3]. This ID is assigned to the most significant four bits of a key. That leaves 28 bits for microblocks to use to perform CAM searches. The number of bits required for the unique ID can be changed without affecting any other component of this proposal. However, we believe that it is highly unlikely that more than 16 (maximum number represented using 4 bits) microblocks using CAMs independently will run on a single microengine in the future. We also believe that a 28 bit-wide key is sufficient to perform searches.

The solution to the second problem is to always yield a free CAM entry when there is a CAM miss. It should be noted that in *folding*, you do not need the least recently used entry on a CAM miss; you only need one free CAM entry that no other thread is using[1]. A free entry can be identified by maintaining a simple bit vector of free CAM entries. We propose use of the following data structures:

1. For every CAM entry, we maintain a reference count representing the number of threads using the CAM entry. The reference count for a CAM entry is incremented when a thread starts using the CAM entry. The reference count is decremented when the thread exits using the CAM entry.
2. We maintain a bit vector, *free_CAM_vector*, with each bit corresponding to a CAM entry. A bit corresponding to a CAM entry is set if and only if the CAM entry is free. A CAM entry is free if and only if the reference count corresponding to the CAM entry is zero.

Identifying a free CAM entry is now as simple as finding a CAM entry with the corresponding bit set in the *free_CAM_vector*. This operation can be easily done using the *ffs* [1] instruction of the microengine. Whenever there is a CAM miss, instead of using the LRU to find a free entry, we find the free CAM entry using the *free_CAM_vector*. We can partition the *free_CAM_vector* among the microblocks running in parallel. For example, if microblock A runs on four threads and microblock B runs on the other four threads, then microblock A always searches for free entries among the first eight entries of the CAM while microblock B searches among the last eight entries of the CAM. This partition can be easily achieved by masking out the bits not corresponding to entries allocated to a microblock.

Assuming that every thread can only use one CAM entry at a time[2], every microblock should be allocated a number of CAM entries at least equal to the number of threads running the microblock. In general, if every thread running a microblock can use a maximum of *n* CAM entries at a time, the microblock needs a number of

[1] When we say that a thread is *using* a CAM entry, we imply that the thread is in one of the "read," "modify" or "write" operations for the data pointed to by the CAM entry.

[2] This is true in all the reference microblocks we have.

CAM entries = *n* * *(number of threads)* running the microblock. Using this method, one can easily see that, every time there is a CAM miss, there will always be a free CAM entry among the CAM entries allocated to the microblock. For example, assume that we allocate each microblock the maximum number of CAM entries (*n* * *(number of threads)*) needed across all the threads running the microblock. Therefore, every time there is a CAM miss, the thread asking for a free entry must not be using all the *n* CAM entries allocated to it. Therefore, less than *n* * *(number of threads)* entries must be in use leaving at least one free CAM entry. If more than *n* * *(number of threads)* are allocated to a microblock, it can rotate the *free_CAM_vector* to use all the CAM entries allocated to it to achieve more caching. However, a minimum of *n* * *(number of threads)* entries are required for the *folding* algorithm to work.

We just described the extensions for the support for *packet multicast and fragmentation* and *CAM Sharing* in the IXA Portability Framework that eases the software development on the IXPs. To further ease the development of the software on the IXPs, we have several advanced features in our development tools and a new programming language. We discuss them in the next sections.

ADVANCED TOOLS FEATURES

The Developer Workbench was introduced three years ago with the first IXP generation, the IXP1200. It has since been upgraded to support the most recent generations of IXP chips, the IXP2400, the IXP2800, and others. The Developer Workbench represents a major step in debug technology, with a patented graphical user interface for the presentation of multiple hardware execution threads, simulation history, and simulation status. It is a tool suite that integrates project software organization, chip configuration, microcode assembly, C compilation, linking, packet generation, cycle-accurate simulation, source-level debug, queue status monitoring, and simulation thread history. The tool suite has established a reputation as the leading development environment for network processors.

So far, our tools have provided a view of the software executing with no special features added to take into account the application domain. However, with network processors, the objective is clear. The application is working on packets or cells, forwarding data in a network environment. With this in mind, we are making the tools more application friendly, to make debugging more *packet-centric*.

Architecture Tool

When deciding which IXP chip to use, or whether to even select one of Intel's IXP processors for a network processor-based system, there are many issues to consider. Will the application code fit in the microengine? What processing stage partitioning will likely meet the line rate requirements? Will the desired memory accesses per packet fit within the bandwidths supported by the chip? How many microengines will be needed to support the desired line rate? These questions and many more can be answered by the new IXP Architecture Tool.

Figure 2 shows the packet processing stage block diagram for a POS OC-48 design. Using the IXP Architecture Tool, the user can lay out the partitioning of the design, specify data structures and task flow, and finally run analysis to determine whether the design "fits."

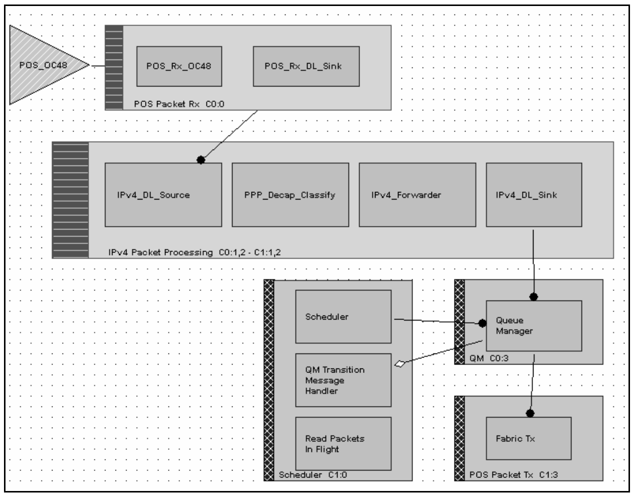


Figure 2: OC-48 design using the architecture tool

The Architecture Tool allows one to determine the feasibility of a design, before embarking on the full-blown project. With the major partitioning done, programmers can be assigned in parallel to work on coding the various blocks. In addition, the Architecture Tool views of code partitioning and data structures make it easy to determine where previous code can be reused (for example, microblocks from the IXA Software Framework), and where new code needs to be written.

Once feasibility has been proven, the Architecture Tool can be used to create a skeletal *Developer Workbench* project. With this as the implementation starting point, code can be written, compiled, and readied for debug.

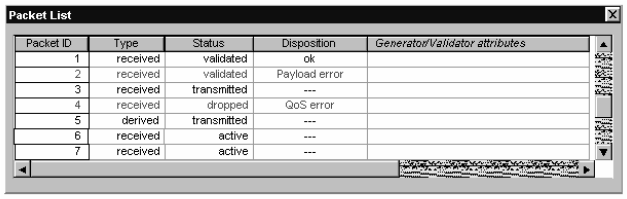
Packet Generation

We have developed and added a new packet generator to the Developer Workbench, called *PacketGen*. PacketGen provides the capability to continually source and validate packets, according to the specification of plug-in protocol modules that define protocol types and a traffic specification of packet grouping into flows. The flows correspond to connections in ATM or TCP/IP. The headers can be layered for a flow in any order, and flows can be bundled within other flows. Sequencing algorithms include round-robin, token-bucket, and random order. There are many choices for setting the rate on a flow, from simple peak rate to the detailed parameters of an ATM TM4.1-based flow specification.

Packet Status View

Previously, finding a packet in a simulation was a laborious effort unless you were the actual designer of the code, in which case you would have known where to set the breakpoints to stop execution at key packet processing milestones. Even so, debugging and tuning the design involved many reruns of simulation and many attempts to capture data that would lead to the cause of a problem. It was far more difficult to debug another's code. To speed up the debugging and performance tuning tasks of the project, a packet profiler has been added to the Developer Workbench. The packet profiler provides the capability of tracing packets from packet generation, through many events in the IXP chip simulation, all the way to validation of output packets after transmit.

Figure 3 shows the new packet status view. This view is the starting point for packet-centric debugging. It shows the state of a packet, such as whether it has been received into the chip, transmitted, derived from another packet, or dropped. If it was dropped, the reason is displayed. If validation caught an error, such as an invalid header format, the symptom information is displayed.


Packet List

Packet ID	Type	Status	Disposition	*Generator/Validator attributes*
1	received	validated	ok	
2	received	validated	Payload error	
3	received	transmitted	---	
4	received	dropped	QoS error	
5	derived	transmitted	---	
6	received	active	---	
7	received	active	---	

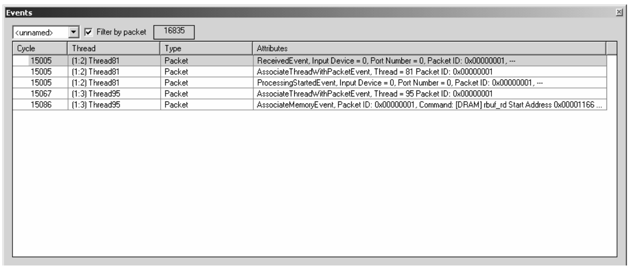
Figure 3: Packet status view

When the user sees a problem in the packet status view, a line can be highlighted to flag the "packet of interest." At this time, a right click will take the user to the code associated with a major event, such as received (first code to work on that packet), or transmitted (last code to work on that packet).

Event View

A packet may actually be in the chip for several thousand cycles. Having two major events (transmitted and received) four thousand cycles apart narrows it down somewhat, but there is still much tracing to do.

The *filtered event view* is shown in Figure 4. Normally, this is a very big list of all history events, including memory reads/write, processing started on a piece of code by a thread, processing started on a microblock, packets received and transmitted, and other events. By selecting the *Filter by Packet* choice in the events dialog box, the number of events is now trimmed to just show those events involving the "packet of interest."


Events

<unnamed> [✓ Filter by packet] 16835

Cycle	Thread	Type	Attributes
15005	(1:2) Thread81	Packet	ReceivedEvent, Input Device = 0, Port Number = 0, Packet ID: 0x00000001, ---
15005	(1:2) Thread81	Packet	AssociateThreadWithPacketEvent, Thread = 81 Packet ID: 0x00000001
15005	(1:2) Thread81	Packet	ProcessingStartedEvent, Input Device = 0, Port Number = 0, Packet ID: 0x00000001, ---
15067	(1:3) Thread95	Packet	AssociateThreadWithPacketEvent, Thread = 95 Packet ID: 0x00000001
15086	(1:3) Thread95	Packet	AssociateMemoryEvent, Packet ID: 0x00000001, Command: [DRAM] rbuf_rd Start Address 0x00001166 ...

Figure 4: Packet events–filtered

Now the user can right click to go to the exact places in the code associated with these events, and can then verify major checkpoints along the packet's life as it is being processed by the application.

Packet Dataflow View

Previously, the thread history view was the primary debug window for watching the behavior of threads. This provides horizontal lines, one per thread, for up to 128 threads. The user could scan backward and forward in time and observe memory access events from request to completion. Also, the user could place labels in code to signify important code points and select them for display on the thread line of the thread history window.

Taking the history window paradigm further, Figure 5 illustrates the packet dataflow view concept. Instead of threads running horizontally in the window, there are packets displayed. In addition to memory references, the data values of the memory reference are shown. Finally, the microblock partitioning is front and center, showing the flow of code for a given packet.

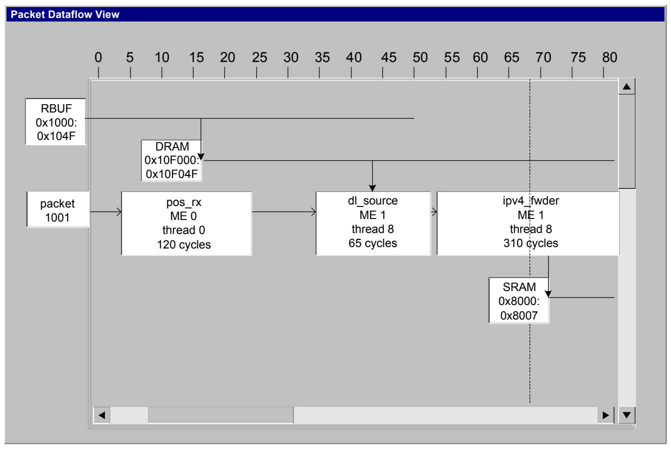


Figure 5: Packet dataflow view

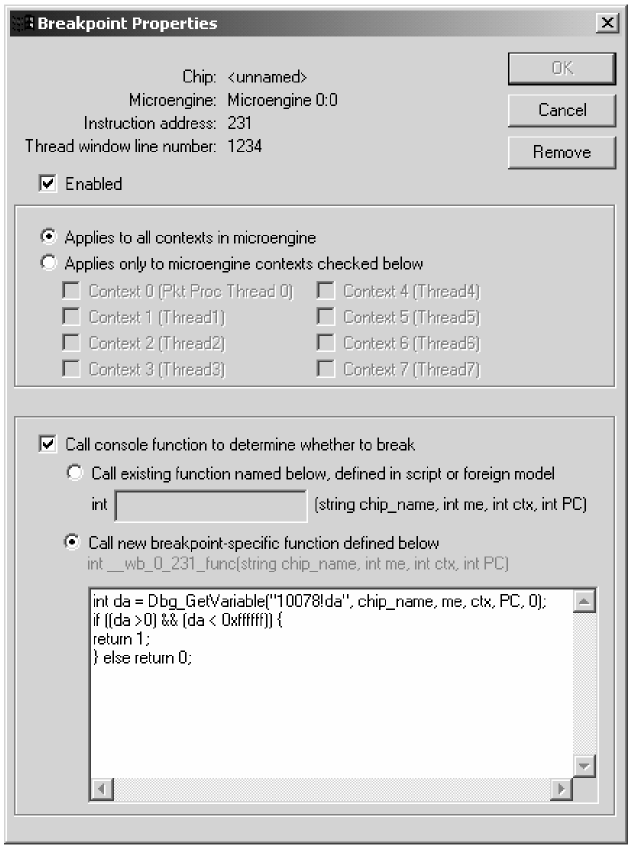
The interaction of more than one packet can also be shown, such as access to the shared data in critical sections.

Using the packet dataflow view, the user can see at a glance the actual design running as envisioned in the Architecture Tool, with tasks and data structures. Erroneous data in data structures, such as bad packet data, incorrect table entry, or wrong inter-thread communication will be shown in this view, thus eliminating many steps of finding and looking up data values.

Conditional Breakpoint

Previously, only unconditional breakpoints could be placed on specific lines of code. The GUI supported a selection of microengine contexts for the breakpoint to be applied. This typical debug procedure was a time-consuming process of manually stepping through breakpoints until the desired condition occurred. Figure 6 shows the *breakpoint editor enhancement*, which provides a true conditional breakpoint capability.

The editor provides a function wrapper for the breakpoint. From within the function, the user can access and set all simulation states, including microengine register variables and memory values. It is even more powerful, in that registered console functions for any foreign model can also be called. It acts just like a function defined in the command line C-Interpreter. In this example, the breakpoint function accesses the IP destination address variable, tests whether it is within a range of values, and breaks only if the condition is satisfied.


Breakpoint Properties

Chip: <unnamed>
Microengine: Microengine 0:0
Instruction address: 231
Thread window line number: 1234

[✓] Enabled

(●) Applies to all contexts in microengine
( ) Applies only to microengine contexts checked below
[ ] Context 0 (Pkt Proc Thread 0) [ ] Context 4 (Thread4)
[ ] Context 1 (Thread1) [ ] Context 5 (Thread5)
[ ] Context 2 (Thread2) [ ] Context 6 (Thread6)
[ ] Context 3 (Thread3) [ ] Context 7 (Thread7)

[✓] Call console function to determine whether to break
( ) Call existing function named below, defined in script or foreign model
int (string chip_name, int me, int ctx, int PC)
(●) Call new breakpoint-specific function defined below
int __wb_0_231_func(string chip_name, int me, int ctx, int PC)

```
int da = Dbg_GetVariable("10078!da", chip_name, me, ctx, PC, 0);
if ((da >0) && (da < 0xffffff)) {
return 1;
} else return 0;
```

Figure 6: Breakpoint properties dialog

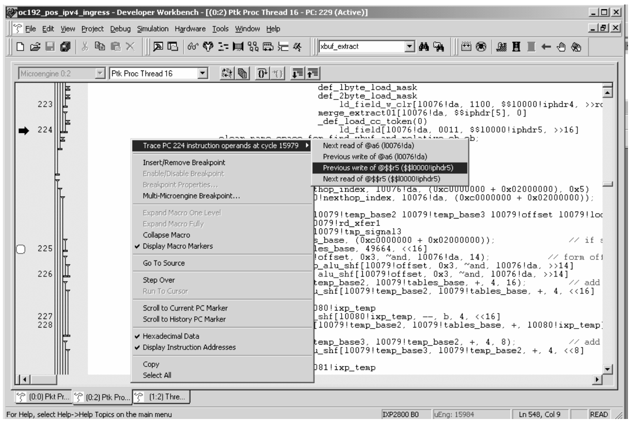
The conditional breakpoint can be used to mark packet events as well, such as the creation of derived packets for multicast, the notification for a dropped packet, or it can tag that a packet processing stage has been entered. These events will show up in the packet events view. Major events, such as packet derived or dropped, will show up in the Packet Status View.

As we described earlier, when the event has appeared in packet status or events view, tracing a bug is a simple matter of highlighting the "packet of interest" in either of these two windows, and right clicking to go to the code that caused the event.

Instruction Operand Trace

Previously, when simulation had stopped, the user would scan through the code in *thread list view*, hovering over variables to find one that was not "right." Sometimes this could be accelerated by using the editor's *find* feature, with the guess that the previous setting of a register was in the up direction and in the same context and microengine. To speed up the tracing of variable values, the instruction operand trace feature is added to the Developer Workbench.

Figure 7 shows the use of instruction operand tracing in the thread list view.


oc192_pos_ipv4_ingress - Developer Workbench - [(0:2) Ptk Proc Thread 16 - PC: 229 (Active)]
File Edit View Project Debug Simulation Hardware Tools Window Help
xbuf_extract
Microengine 0:2
Ptk Proc Thread 16
Trace PC 224 instruction operands at cycle 15979
Next read of @a6 (l0076!da)
Previous write of @a6 (l0076!da)
Previous write of @$$r5 ($$l0000!iphdr5)
Next read of @$$r5 ($$l0000!iphdr5)
Insert/Remove Breakpoint
Enable/Disable Breakpoint
Breakpoint Properties...
Multi-Microengine Breakpoint...
Expand Macro One Level
Expand Macro Fully
Collapse Macro
Display Macro Markers
Go To Source
Step Over
Run To Cursor
Scroll to Current PC Marker
Scroll to History PC Marker
Hexadecimal Data
Display Instruction Addresses
Copy
Select All
(0:0) Pkt Pr...
(0:2) Ptk Pro...
(1:2) Thre...
For Help, select Help->Help Topics on the main menu
IXP2800 B0
uEng: 15984
Ln 548, Col 9
READ

```
223                                 def_1byte_load_mask
                                    def_2byte_load_mask
                                        ld_field_w_clr[l0076!da, 1100, $$l0000!iphdr4, >>rc
                                    merge_extract01[l0076!da, $$iphdr[5], 0]
                                    _def_load_cc_token(0)
224                                     ld_field[l0076!da, 0011, $$l0000!iphdr5, >>16]
                       clear name space for find xbuf and relative sb eb;
                                    thop_index, l0076!da, (0xc0000000 + 0x02000000), 0x5)
                                    0!nexthop_index, l0076!da, (0xc0000000 + 0x02000000));

                                    l0079!temp_base2 l0079!temp_base3 l0079!offset l0079!loc
                                    0079!rd_xfer1
                                    l0079!tmp_signal3
225                                 s_base, (0xc0000000 + 0x02000000));              // if s
                                    les_base, 49664, <<16]
                                    !offset, 0x3, ~and, l0076!da, 14);             // form off
226                                 p_alu_shf[l0079!offset, 0x3, ~and, l0076!da, >>14]
                                     alu_shf[l0079!offset, 0x3, ~and, l0076!da, >>14]
                                    temp_base2, l0079!tables_base, +, 4, 16);           // add
                                    u_shf[l0079!temp_base2, l0079!tables_base, +, 4, <<16]

227                                 080!ixp_temp
228                                 _shf[l0080!ixp_temp, --, b, 4, <<16]
                                    [l0079!temp_base2, l0079!tables_base, +, l0080!ixp_temp]

                                    temp_base3, l0079!temp_base2, +, 4, 8);              // add
                                    u_shf[l0079!temp_base3, l0079!temp_base2, +, 4, <<8]

                                    081!ixp_temp
```

Figure 7: Instruction operand tracing

When the user selects the backward trace to the previous write of an operand, the thread list goes to that code line, and the current "cycle of interest" is changed for all event and history views. If another microengine wrote the operand (e.g., a next-neighbor write), the thread list view will bring up the thread in the other microengine and scroll to the line of code that initiated the write. This saves a couple of steps in the search; plus, it takes the user quickly to the cause or the result.

Considering the hundreds of times a user tries to trace operand values during design test and validation, the cumulative effect of instruction operand trace feature results in a significant time saving.

Tool Enhancements Summary

The tool enhancements provide a major step forward in network processor tool technology. Collectively, they shift the focus of development toward the application domain. They provide a packet-centric approach to the design and debug of network processor applications. In this paper we presented the following tool enhancements:

- *Architecture Tool*, for top-level partitioning and feasibility analysis.
- *Advanced Packet Generation*, for continuous generation and validation according to extensible protocol plug-ins.
- *Advanced Packet Profiler*, with packet list view summarizing packet status and providing hooks to other graphics views.
- *Comprehensive Event History*, with filtering and instant jump to associated code.
- *Packet Dataflow View*, extending the Packet Profiler with presentation of packet code flow and associated data structures.
- *Conditional Breakpoint*, with full access to simulation states and simulator console functions.
- *Instruction Operand Tracing*, with fast jump forward or backward in simulation history.

So far, we have discussed incremental extensions or enhancements to the IXA software framework and development tools. However, we have also been investigating some long-term directions in simplifying the software development by developing a next-generation, packet processing language.

NEXT-GENERATION PACKET PROCESSING LANGUAGES

Why Build A New Language?

Network processing (NP) systems are designed to meet two, often conflicting, requirements: support of a large number of high-bandwidth links, and hence large system throughputs and, at the same time, offer a wide range of services. To meet these requirements simultaneously, NPs support mechanisms such as multiple processor cores per chip and multiple hardware contexts per processor core that enable them to process network packets at high rates.

Currently, the languages used for programming these systems expose much of the hardware details to the programmer. Hence, to develop high-performance packet processing applications on such systems, programmers are required to become intimately familiar with the hardware and develop hand-tuned code for managing NP resources.

One solution to the above problem is to take a general-purpose language such as C and enhance it with new constructs and programmer-specified annotations to allow representation of independent modules and their interactions. In this approach, the compiler uses the information gleaned from the use of new constructs and program annotations to automatically map a sequential program onto multiple processing cores and threads. A programmer is still responsible to map various data structures in different parts of the memory hierarchy. IXP C is such an extension to the standard C language that has been described elsewhere [7].

In this paper, we describe yet another approach that hides almost all the hardware details by taking an existing general-purpose programming language and extending it with domain-specific features, to enable both the programmer and compiler to build efficient network systems. Here we describe the vision of such a language called Baker.

What is Baker?

Baker is the programming language for a network application development environment currently being researched, called *Shangri-la*. Baker is a domain-specific programming language in that, although it is based on C, it has many packet-processing specific features added to it. These features not only make the job of the programmer easier but they also narrow the scope of possible applications the compiler needs to handle, and expose critical information enabling the compiler to make informed decisions and generate efficient binaries.

Domain-Specific Features

The domain of packet processing has several key features that differentiate it from other problem domains. In the packet-processing domain, the code and data are usually highly independent. NPs have been designed with multiple cores and multiple hardware threads in order to take advantage of this independence, and to be able to execute much of the code in parallel. This results in one of the most difficult jobs of the developer: partitioning the program among the cores and threads.

Baker exposes this concurrency of the data by explicitly representing packets in the language, and exposes concurrency in code through a data flow model.

Packets

The majority of the work done in NP systems involves working with the data and metadata of packets. Baker treats packets as first-class types in order to identify to the compiler data that is independent, and hence parallelizable. This enables the compiler to optimize code working with this data. It also gives the programmer a higher level view of them, simplifying the code.

Data-Flow

One of the challenges of programming multicore multithreaded systems is determining how to partition the application in such a way that it can be executed over the cores and threads to take advantage of the highly threaded hardware and still keep the code reusable. Much research has gone into compilers that can automatically partition programs written in general-purpose languages, but this is generally ineffective in this domain because these compilers must make conservative assumptions about the independence of the code and data. Baker's approach to this problem is to use a data flow model. In this model, a programmer can make it known to the compiler which pieces of the application are reasonably independent and therefore can be run in parallel. With the addition of knowledge of packets and their independence, the compiler can then lay these pieces out on the cores and threads as it sees fit.

In a data flow model, many functional units, or *actors*, are linked together to form a greater application. These actors process their input and produce outputs, analogous to the *pipe and filter model* in a UNIX shell. Baker uses the concepts of Packet Processing Functions (PPFs), and *channels* for describing the data flow, which are heavily influenced by the language Click [4]. The programmer writes several reasonably independent PPFs to do the work of a network application and connects the PPFs via channels in a module. A module is a collection of PPFs and channels. A PPF is an actor in the data flow model and contains the data and functionality for a small piece of a network application. PPFs can have any number of inputs or outputs that are later connected to other PPFs by channels. Channels are typed, unidirectional interconnects that may or may not be synchronous. Because the definition of a channel is very open, the compiler is free to replace them with whatever construct it deems optimal for a particular system.

Figure 8 illustrates a simple L3 switching module described in a data flow model, containing three PPFs, and the following code snippet represents how its definition would be written in Baker.

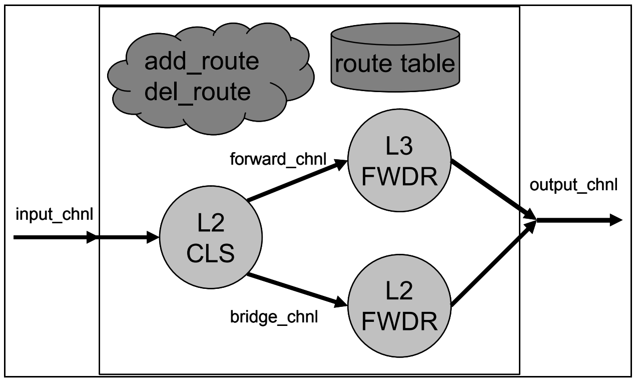


Figure 8: An L3 switch module

```
// L3 switch module
module L3Switch {
  ppf L2Cls;  //forward declarations of
  ppf L2Fwdr; //required PPFs
  ppf L3Fwdr;

  channels {
    input packet input_chnl;
    output packet output_chnl;
  }
  wiring {
    //equate this module's external channel
    //endpoints to internal PPFs'
    input_chnl = L2Cls.input_chnl;
    output_chnl = L2Fwdr.output_chnl;
    output_chnl = L3Fwdr.output_chnl;

    //bind internal PPFs channel endpoints
    L2Cls.bridge_chnl -> L2Fwdr.input_chnl;
    L2Cls.forward_chnl ->L3Fwdr.input_chnl;
  }

  //module’s data
  route_table_type route_table;

  //module’s interface
  void add_route(route_t r);
  void del_route(route_t r);
};
```

In this example, the *L3Switch* declares the PPFs that it contains and binds their channel endpoints together. This binding defines the processing pipeline for this module. The *L3Switch* also exposes two channel endpoints of its own. These endpoints are equated to endpoints of internal PPFs, allowing the module to be used by another developer without knowing its internal details. Any packets sent to the module’s input channel will be sent to the L2Cls PPF. This wiring concept is borrowed from the nesC [5] language developed for TinyOS [6].

Consequences of Partitioning a Baker Application

Once the application is written, the Baker compiler uses knowledge about the packets and data flow (actors and channels) to partition the code among the processors and threads of the NP.

Many PPFs are capable of being replicated and run on many cores simultaneously as long as the data that they are working on is reasonably independent. Baker assumes that all packets are completely independent, and thus all PPFs may be replicated. If packets worked on by two or more PPFs are interdependent, then a programmer must specify ordering criteria for the packets in the definition of the input channel of the PPF.

Given that the code of the network application may be run in parallel in a way that is completely determined by the compiler, the programmer must understand at all times that any code may be run by any number of threads simultaneously. Therefore, a Baker programmer must take precautions, such as using locks, to protect any data that may be shared to avoid problems.

Portability

One advantage of a domain-specific language such as Baker, is that it can aid in the portability of applications. Specifically, by abstracting the key aspects of the hardware, the programmer no longer becomes concerned with the exact processor configuration in the NP.

To this end, Baker presents an abstract hardware model to the programmer to simplify both the programming and porting of applications. The basic model is one that is multithreaded with a flat, shared memory. Although the model is multithreaded, the programmer has no control over or knowledge of the threads. This lack of control is important to avoid burdening the compiler with the task of automatically synchronizing access to shared data. Rather than try to solve the hard problem of building a compiler that can analyze code to automatically find any synchronization issues on its own, Baker leaves that work to the programmer. Because the programmer knows the system is multithreaded, all global or shared data must be protected via locks.

The flat memory model presented to the programmer simplifies coding as the developer no longer has to place variables in one of any number of memory types and locations, or write the accompanying specialized accessing code. The compiler can automate the task of mapping data structures to memory types, for example, using knowledge such as which memory accesses are for packets and which are not.

As an example of portability, let’s consider how Baker enables programmers to work with packets without dealing with different memory types or packet buffer data structures. Baker allows the programmer to define protocol header templates, and then access packet data using protocol fields and several built-in functions. A protocol header template is much like a struct in C, except that there are no types to each field, only sizes, and some special features have been added to handle special cases, such as optional fields.

An example of a protocol definition for EthernetSNAP, and the subsequent packet access through that protocol is given below:

```
// The LLC and SNAP subfield
field LLCSNAP {
  DSAP : 8;
  SSAP : 8;
  Control : 8;
  Vendor : 24;
  LocalCode : 16;
};
protocol Ethernet2 {
  dest : 24;
  src  : 24;
  len  : 16;
  snap : anyof {
    { len > 1500 }: 0;
    default: LLCSNAP;
  }
  demux { (len<1500)?len:14 };
};
```

Here we have defined two structures, one a protocol and the other a sub-field of a protocol. Elements of each are given a name and a length in bits. The ordering is important, as each field is found in the packet by the total number of bits used up before it. The compiler generates the correct code to access the bits in the actual packet when the protocol is used by the programmer, no matter how the packet is represented in memory. Protocols are allowed to define pseudo-fields, which do not affect the placement of the regular fields, but give a tool to the protocol developer to ease the burden on the programmers using it. The *anyof* directive is a way for a protocol to have optional fields. Here, the field *snap* takes on the properties of either a zero length field or of an LLCSNAP, depending on the evaluation of the cases of the *anyof*. The *demux* pseudo-field is a required field for all protocols which allows the compiler to generate the code necessary to find the next encapsulated protocol. Depending on the value of *len*, the next protocol header can be found either *len* bytes or 14 bytes after the beginning of this header.

```
//example function using the above protocol
void process(Ethernet2_packet_t* p)
{
  IPv4_packet_t* ip;
  if (p->len < 1500)
    processSnap(p)

  ip = packet_decap(p);
  if (ip->version == 6)
    processIPv6(ip)
  else
    processIPv4(ip)
}
```

Notice the new types, *Ethernet2_packet_t* and *IPv4_packet_t*. Once the compiler has parsed a protocol definition, it creates new types for them, which are subtypes of the built-in packet type. The fields are accessed just as the fields of a struct would be, which makes coding very simple, and hides any implementation details. The packet_decap function uses the required demux field to find the next header and returns a pointer to the protocol found there. Notice that whether or not there was a SNAP header, the code to decapsulate is exactly the same.

Given that no details are provided to the programmer, the compiler is free to make any optimizations to the handling of packets and data. And if anything changes, or the code is targeted to a system that handles packets completely differently, then all that is required is a simple recompile.

CONCLUSION

We first identify the limitations of the current IXA software framework and development tools, and then describe the incremental enhancements as well as some breakthrough research in providing much better tools to an IXP developer. We describe simple extensions to the framework to overcome these limitations without affecting performance. These extensions facilitate the development of new applications (such as multicast, fragmentation) using the IXPs while supporting seamless portability between very high-end and low-end IXPs.

In addition, we describe features of the advanced tools. Collectively these features shift the user's focus toward the application domain of packet-centric processing, and away from the hardware details. Rather than having to remember the relationship between application code and microengine/thread assignments, the user can now focus on the progress of test packets and major application events. The user can instrument code and create special events that can also be traced.

The packet-centric approach enables instrumentation of code without inserting hardware-specific hooks. Therefore, the instrumentation portion itself is also portable. The new packet-centric events and dataflow views also help in debugging the same design that has been ported across IXP versions.

Finally, we discussed future directions in packet processing languages that provide benefits to both programmer and compiler writer, and gave a brief overview of how to realize some of these directions in the Baker language. Only a few of the advantages and tools it provides for easing the development of packet processing applications are discussed here. Although much research remains to be done on Baker and its compiler environment, Shangri-la, we already see promising results in related technologies such as the IXP-C [7].

ACKNOWLEDGMENTS

The work presented here resulted from the contribution of many members of our team. Authors are especially grateful to Uday Naik, Prashant Chandra, Erik J. Johnson, and Eric Walker for in-depth discussions and original technical contributions.

REFERENCES

[1] Matthew Adiletta et al., "The Next Generation of Intel IXP Network Processors," *Intel Technology Journal*, Volume 6, Issue 03, August 15, 2002, pp. 6-18.

[2] *Intel® IXP2400/2800 Software Development Toolkit* Version 3.5, *Intel® IXP2400/2800 Network Processors Development Tools User's Guide*, September 2003.

[3] Uday Naik et al, "IXA Portability Framework: Preserving Software Investment in Network Processor Applications," *Intel Technology Journal*, Volume 6, Issue 03, August 15, 2002, pp. 50-60.

[4] E. Kohler, et. al., "The Click Modular Router," *Technical Report Laboratory for Computer Science*, MIT, 2000.

[5] David Gay, Phil Levis, Rob von Behren, Matt Welsh, Eric Brewer, and David Culler, "The nesC Language: A Holistic Approach to Networked Embedded Systems," in *Proceedings of Programming Language Design and Implementation* (PLDI) 2003, June 2003.

[6] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister, "System Architecture Directions for Networked Sensors," *Architectural Support for Programming Languages and Operating Systems*, pp. 93–104, 2000.

[7] Intel Corporation, "Introduction to Auto-Partitioning Programming Model," *Literature number 254114-001 at http://www.intel.com/design/network/papers/254114.htm*

AUTHORS' BIOGRAPHIES

Stephen D. Goglin is a network software engineer in the Communications Technology Lab of CTG at Intel Corporation. He received his M.Sc. degree in Computer Science from the University of Victoria in 1999. Stephen's interest includes graph theory, code generation, and his wife and two children. His e-mail is stephen.d.goglin at intel.com.

Donald Hooper is a senior software architect in the Network Processor Group. He has led many projects including Logic Synthesis, Video Servers, MPEG 2 and DAVIC Standards, IXP1200 Software Tools and Libraries, IXP2800 Proof of Concept Designs, and NPG Coding Standards. His professional interests include networking, artificial intelligence, and object-oriented languages. He attended four colleges with cumulative B.S.E.E. degree credits finishing at UCLA. He resides in Shrewsbury, Massachusetts, and can be reached via e-mail at donald.hooper at intel.com.

Alok Kumar is a software architect in the Software Architecture Group of the CIG GTO at Intel Corporation. He has worked on new algorithms for IPv6 forwarding, packet classification and software framework for IXP2xxx. His interests are in the areas of high-speed programmable routers, quality of service, algorithms and computer graphics. He received his B.Tech degree in Computer Science from the Indian Institute of Technology, Delhi, in 1999, and his M.S. degree in Computer Science from the University of Texas at Austin in 2001. His e-mail is alok.kumar at intel.com.

Raj Yavatkar is currently the chief software architect for the Internet Exchange Architecture at the Intel Communications Group. Raj Yavatkar received his Ph.D. in Computer Science from Purdue University in 1989. He is a co-author of the book "Inside the Internet's Resource reSerVation Protocol (RSVP)" published by John Wiley in 1998. He currently serves on the editorial board of the IEEE Network magazine. His e-mail is raj.yavatkar at intel.com.

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.

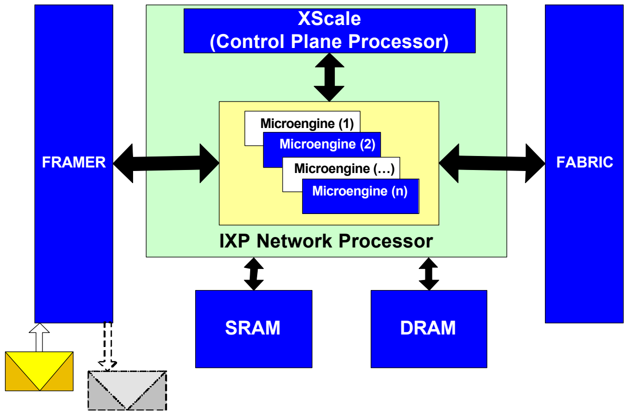


Figure 1: Packet processing pipeline

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).

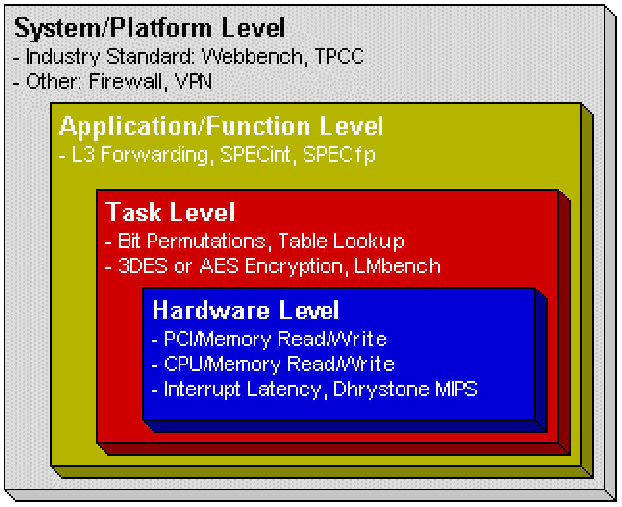

System/Platform Level
- Industry Standard: Webbench, TPCC
- Other: Firewall, VPN
Application/Function Level
- L3 Forwarding, SPECint, SPECfp
Task Level
- Bit Permutations, Table Lookup
- 3DES or AES Encryption, LMbench
Hardware Level
- PCI/Memory Read/Write
- CPU/Memory Read/Write
- Interrupt Latency, Dhrystone MIPS

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:

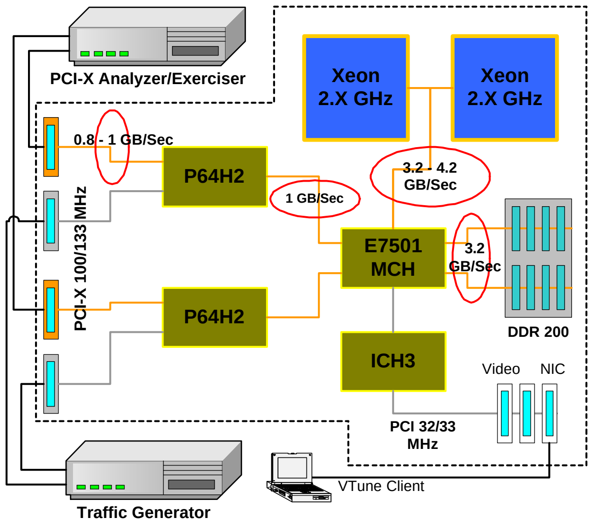


Figure 3: Standard IA32 network test platform

- *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.

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

Also,

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

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.

$$\textit{Bus Utilization (\%)} = \frac{\textit{Busy Clocks}}{\textit{Total Clocks}} \quad x\ 100$$

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.

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

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.

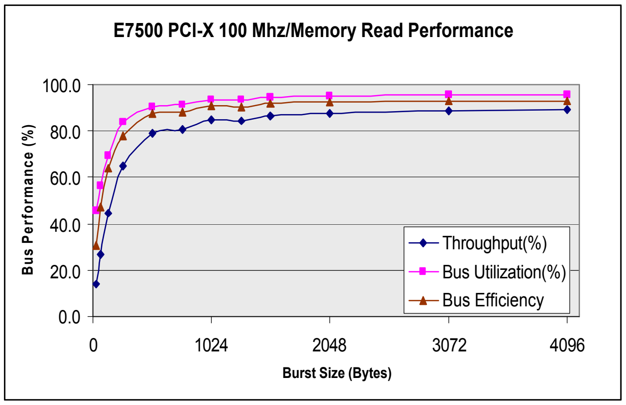


Figure 4: PCI-X Memory read block tests

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.

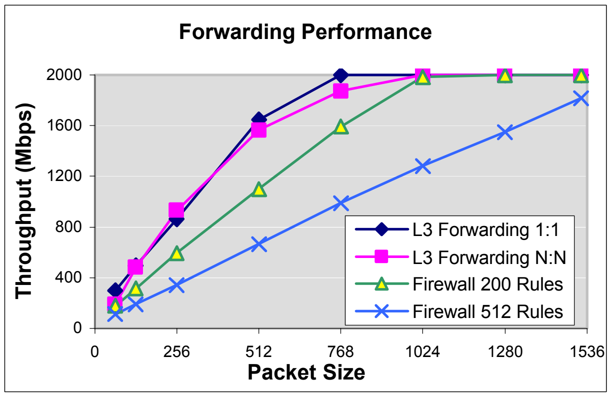


Figure 5: Single processor forwarding

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*.

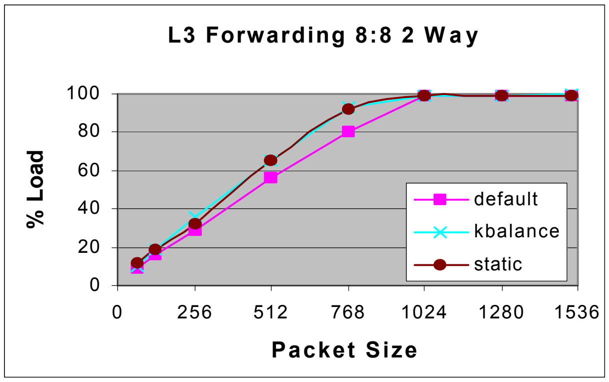


Figure 6: Dual processor forwarding

- 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.

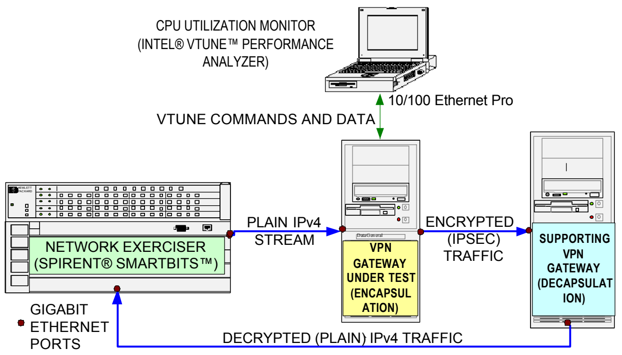

CPU UTILIZATION MONITOR (INTEL® VTUNE™ PERFORMANCE ANALYZER)
10/100 Ethernet Pro
VTUNE COMMANDS AND DATA
NETWORK EXERCISER (SPIRENT® SMARTBITS™)
PLAIN IPv4 STREAM
VPN GATEWAY UNDER TEST (ENCAPSULATION)
ENCRYPTED (IPSEC) TRAFFIC
SUPPORTING VPN GATEWAY (DECAPSULATION)
GIGABIT ETHERNET PORTS
DECRYPTED (PLAIN) IPv4 TRAFFIC

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.

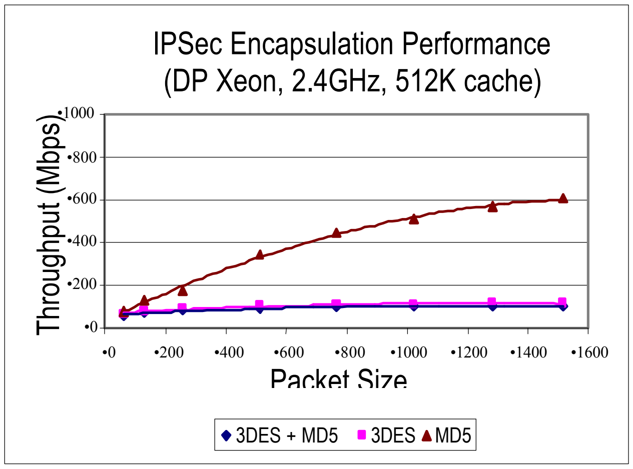


Figure 8: IPSec encapsulation performance vs. packet size for different 3DES/MD5 options

- 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.

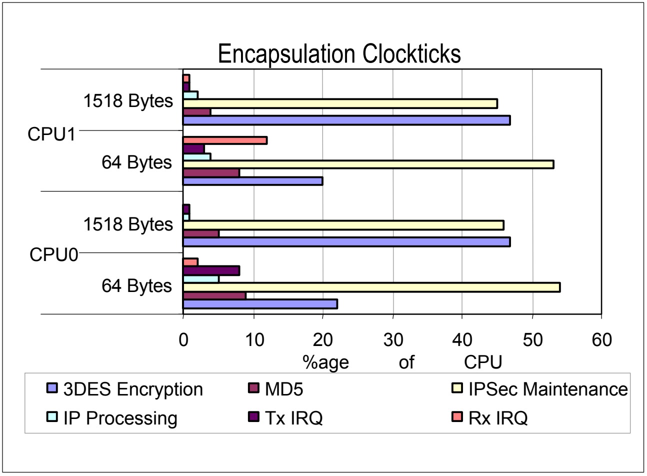


Figure 9: CPU utilization distribution among various functions in IPSec

- *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 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.

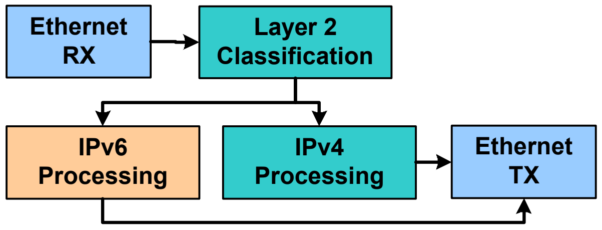


Figure 10: IP benchmark software architecture

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 table[2] 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

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

® 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.

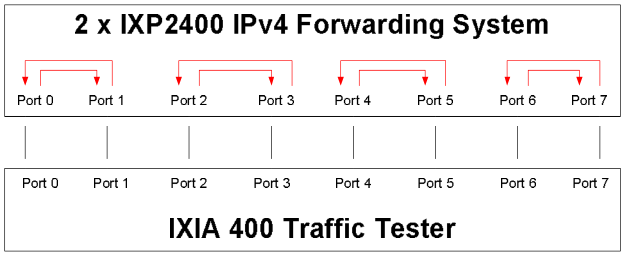


Figure 11: Traffic configuration

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.

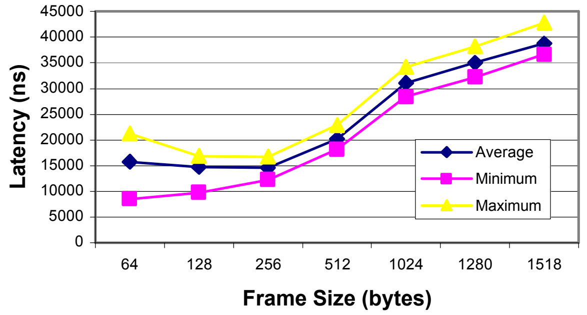


Figure 12: IPv4 Latency at 100% throughput rate

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.

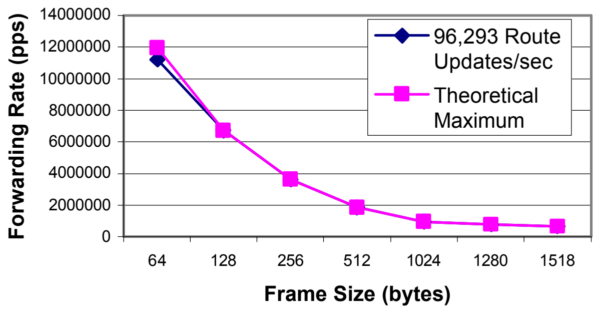


Figure 13: Forwarding rate with concurrent route updates

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.

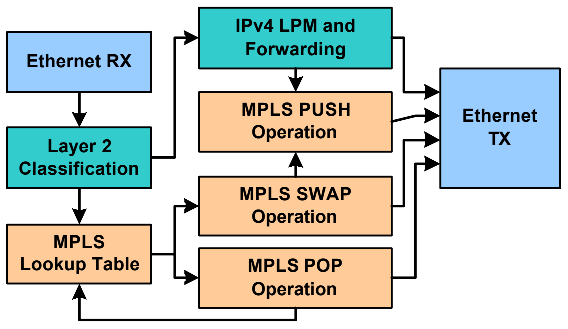


Figure 14: MPLS processing system architecture

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.

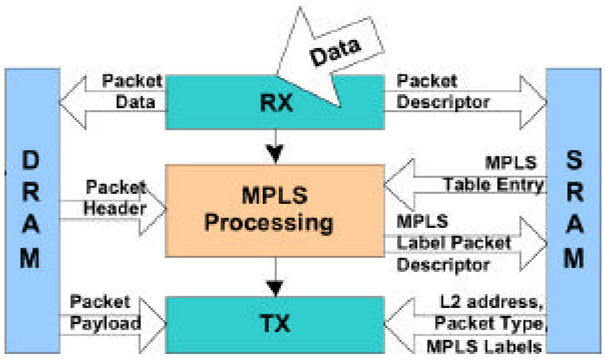


Figure 15: MPLS processing data flow

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.

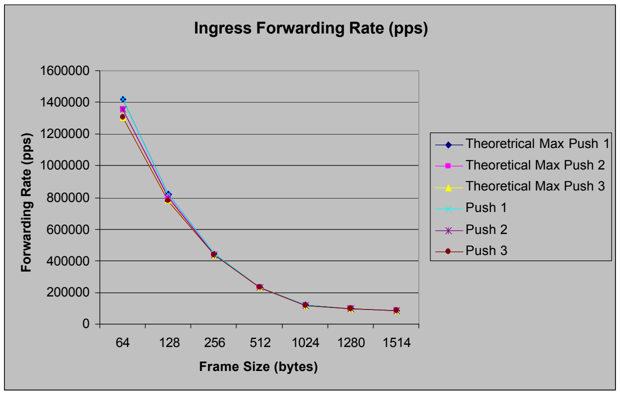


Figure 16: Forwarding rate for all packet sizes in ingress LER test

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].Insolvibile, 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.

Enterprise Edge Convergence: Packet Processing and Computing Silicon in the Data Center

Matthew Adiletta, Intel Communications Group, Intel Corporation
John Beck, Intel Communications Group, Intel Corporation
Doug Carrigan, Intel Communications Group, Intel Corporation
Mark Rosenbluth, Intel Communications Group, Intel Corporation
Bill Tiso, Intel Communications Group, Intel Corporation
Frank Hady, Technology and Manufacturing Group, Intel Corporation

Index words: enterprise, edge, communication processors, network processors, application processors, architecture, microprocessors, bladed servers, servers, load balancers, Layer 7, SSL, firewalls, security, storage, TCP, TOE, Xeon, IXP2xxx, virtual, convergence, media servers, VPN

ABSTRACT

This paper describes the need for enhancing data availability and integrity and reducing the cost of ownership of computing in the enterprise. The "front-edge" and "back-edge" are discussed and the ideal mid-sized enterprise highlighted. We introduce the usefulness of a packet-processing capability in the enterprise and of an architectural approach that converges the complementary capabilities of IXP network processors and the Intel Architecture (IA) in a coherent system.

INTRODUCTION

Enterprise Information Technology (IT) managers are seeking improved data availability and integrity and reduced cost of ownership. Data availability refers to the need for compute resources and network resources to be useable and functioning at all times. Data integrity addresses the issue of consumers of data having confidence that the data they are using has been unaltered by third parties. Cost of ownership is the inherent expense of operating a given enterprise system.

The *Enterprise Edge* represents the gateway between clients and servers–the point where decisions must be made as to who gets access to critical data center resources and under what conditions this access is given. This is the ideal point to implement management schemes that will determine how cost-effectively and how securely the enterprise IT infrastructure will operate. This infrastructure and the management policies implemented here will dictate the productivity of virtually all workflow processes in the enterprise.

Reducing the cost of ownership is based on the belief in the inherent value of converging multiple functions into a single system. Full realization of the purported benefits of convergence will require that the capabilities that exist in individual platforms today (as measured by performance, functionality, and cost) be preserved during the transition to a single unified platform. The integration must also deliver a number of significant new benefits:

- Fewer individual systems must be deployed, requiring less overhead, floor space, power, and cooling, etc. There is also a direct correlation between the number of boxes that must be managed and the size of the required IT staff.
- Fewer different types of boxes are required. A single converged platform that delivers the functionality of multiple individual platforms should result in lower training costs, fewer IT "specialists" and should be easier to manage, maintain, etc.
- A more homogeneous infrastructure will be in place. Many of the difficulties and sources of error in network installation and management are a result of the interdependencies that exist between different systems deployed together.
- There needs to be a unified management domain. The key to realizing operational efficiencies, the ultimate goal, is to provide a single, unified view of

the network and compute environment via the management system.

In examining the enterprise system, there are components that may be optimized. The current definition of the enterprise generally has a network connectivity, going to a farm of web servers, connected through firewalls and load balancers to application servers. The application servers are then attached to file servers that provide requested data to the compute servers. The optimizations that can be created for the enterprise networking infrastructure can generally be categorized as front-edge and back-edge services.

The front-edge generally includes the connection to the Internet, and the subsequent web servers for external users, and Virtual Private Network (VPN) tunnels and load balancers to the Local Area Network (LAN) within the enterprise. The LAN includes application servers, mail servers, printers, etc. The LAN also communicates to the enterprise back-edge.

The back-edge generally includes the path from the LAN to storage. Today this path is generally to Direct Access Storage (DAS). Tomorrow it is expected that Network Attached Storage (NAS) with its virtualization capability will be widely deployed. Tomorrow's back-edge will also enable new capabilities that will enrich the enterprise. Specifically, it will enable high-performance Internet Protocol (IP) media services to the enterprise telephony infrastructure that provide voice, speech, and multimedia to augment and, in the future, replace current PBX solutions.

The eventual conclusion of this evolution is a network and computing infrastructure that is fully autonomic. This infrastructure will be one that can diagnose its own problems and respond with little or no operator intervention; one that automatically senses shifts in demand (or status of available resources) and adjusts its behavior to provide the optimal response; one that identifies critical resources and accelerates access to them; and one that is easy to install, provision, repair, upgrade, and afford. This vision will be built on extremely high-performance computing and packet-processing platforms, whose foundations are the silicon represented by converged Intel Architecture-Intel Exchange Architecture (IA-IXA) silicon.

FRONT-EDGE EVOLUTION STARTING TODAY

The enterprise edge represents the boundary between the pure computing world of the application and database servers found in larger enterprise and service provider data centers and the broad universe of internet or intranet clients that they serve. As shown in Figure 1, the enterprise edge represents the gateway between clients and servers–the point where decisions must be made as to who gets access to critical data center resources and under what conditions this access is given.

This is the ideal point to implement management schemes that determine how cost-effectively and how securely the enterprise IT infrastructure will operate. The infrastructure and the management policies implemented here dictate the productivity of workflow processes within the enterprise. Whether to deliver enhanced manageability or enhanced security, the edge of the enterprise is the focus for much of the innovation in network equipment design. It is at the enterprise edge where much of the complexity of network management exists and a wide variety of platform types are deployed that provide an optimal point to deliver value-added services.

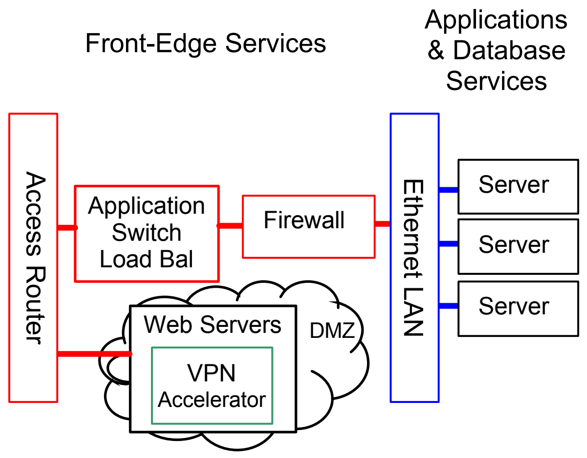


Figure 1: Today's front-edge to LAN and server

Today's Edge Configuration

As the gateway to the enterprise, the enterprise edge is the point at which customers, employees, business partners, and potential attackers first gain access to the information assets of the enterprise. Primary concerns of IT managers are to provide high bandwidth and reliable access to this information while protecting these assets from theft, inadvertent disclosure, and abuse. This is accomplished by carefully managing the traffic that traverses the edge. These gateways serve a number of functions: authenticating users who wish to access the network, protecting (encrypting) data in transit, and defining and enforcing service-level agreements with customers wishing to access the network. An additional goal of these gateways is to manage all of the various access requests in such a way as to optimize utilization of the network and compute resources. Figure 1 is representative of typical current mid-range enterprise edge configurations.

The access router provides connectivity to the Internet, with the ability to manage on the order of 100,000 connections or routes. These routes are typically dynamically managed through route tables via either vector (route hops to destination) or link (connection state to local routers) protocols. Routing tables are dynamically updated and communicated to other routers as network links are added, removed, or become unavailable. A de facto standard protocol is Border Gateway Protocol (BGP), a vector-based protocol.

Multiple configurations of the application switch and firewall are possible depending on the relative needs for performance, security, and load balancing within the edge and web or application servers. The application switch and firewall perform a variety of functions, which are off-loaded from the web and application servers. The firewall supports services such as the following:

- *VPN/IPSec Gateway*. This requires line rate termination of IPSec protocols including full packet encryption, decryption, and authentication using 3DES, AES, SHA1, MD5, and RC4 algorithms. The Virtual Private Network (VPN) then provides the tunnel to the enterprise Local Area Network (LAN).
- *Filtering*. This characterizes Internet Protocol (IP) traffic to identify streams that have permission to access various information, and it can define access to demilitarized zones (DMZ) or intranet resources. Filtering can be done based on mechanisms such as L3-based Access Control Lists (ACLs) or L4-based TCP port blocking.
- Network Address Translation (NAT)

The application switch supports services including the following:

- *TCP termination*. Web servers are deployed in environments where thousands to millions of simultaneous clients submit HTTP requests via independent short-lived TCP connections. These connections must be set up, serviced, and torn down at very high connection rates and must support an aggregate throughput rate in excess of 1 Gigabit/second (eventually 10 Gigabits/second).
- *Intrusion/virus detection*. This requires the ability to analyze entire streams of network data traffic for arbitrary and constantly expanding sets of attack or virus signatures. The data stream is most often segmented across many discontinuous packets. These signatures must be compared against a database of potentially tens of thousands of active rules. This also requires the ability to track protocol state machines for each active connection, looking for improper protocol usage.
- *Traffic manager/Layer 4-7 Load Balancer*. This requires the reassembly of Ethernet packets into TCP streams and subsequent classification of packets using Layer 4 through Layer 7 parameters, including, but not limited to, TCP port numbers, URLs, cookies, user names, application type, etc. Subsequent forwarding/drop priorities are based on these parameters.

Web servers are often established in a DMZ, such that external initiated Internet access requests are able to access data contained within the DMZ, but do not have visibility or access to the internal network or hosts. Both enterprise hosts and appropriate external agents have access to information and servers contained within the DMZ; however, protection protocols prevent external access to internal servers inside the firewall. Servers, such as web or mail servers, are considered to be in a semi-trusted DMZ area, based on optional security or firewall protection present before DMZ access and the level of DMZ isolation to the internal network. Servers contained in the DMZ may also have VPN termination capability.

Traffic that passes through the application switch and firewall now has access to the intranet or enterprise LAN typically via an Ethernet switch, to be serviced by a variety of enterprise data and application servers. HTTP streams are directed to the appropriate web servers. HTTP streams are SSL terminated either via an SSL appliance or capability within the web server. SSL acceleration requires very highly sustained SSL record processing, including public/private key processing using up to 2048-bit keys. Communication intended for other enterprise clients or servers is also routed via the Ethernet switch.

TOMORROW'S EDGE CONFIGURATION

The access router, application switch, and firewall capabilities outlined in the previous section are supported via server-based software or accelerated via stand-alone fixed function appliance products installed at the edge of the network behind the edge router. As outlined, edge functional requirements have spawned a number of different solutions ranging from IPSec and SSL-based VPN gateways through firewalls, NAT proxies, virus scanners and intrusion-detection systems. Because the nature of the problem addressed by these systems often requires very specific processing on all data that traverses the gateway (such as encryption and virus scanning), these applications are very compute intensive. Historically, developers have had to trade off performance (line rate) for functionality when designing these systems, particularly with software-only implementations. Alternatively, non-extendable platform-specific optimizations have been done to get the required level of performance at the expense of time-to-market and solution scalability.

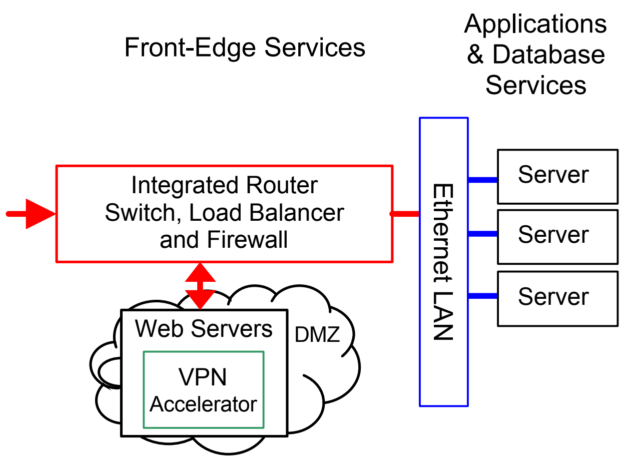

Front-Edge Services
Applications & Database Services
Integrated Router Switch, Load Balancer and Firewall
Ethernet LAN
Server
Server
Server
Web Servers
DMZ
VPN Accelerator

Figure 2: Tomorrow's front-edge configuration

The configuration of future enterprise edge solutions shown in Figure 2 highlights three important enhancements to address the constraints presented by today's edge solutions:

- Multi-function converged hardware acceleration is available to support the application switch, VPN, firewall, load balancing, routing, and TCP termination to enable a full-featured enterprise edge.
- Merged general-purpose (i.e., architecture based on the Intel® Xeon™ processor) compute capability with tightly coupled packet processing (i.e., Intel IXP architecture).
- Reconfigurability and flexibility to support evolving features, standards, and protocols via software programmability.

The consolidation of various appliance and "add-on" boxes into a single device provides inherent advantages to cost, power, footprint area, and on-going IT support. The merged edge solution will provide a platform upon which the various capabilities can be built in a clean consistent way with the ability to separate the "general-purpose" compute functions from "packet processing" compute cycles–with each operation allocated to the appropriate resource.

This configuration not only offloads edge software solutions from the enterprise web and application servers, but provides the necessary processing power to support full-featured capabilities at line rates up to and beyond 10 Gb/sec. Programmable general-purpose and packet processors provide the capability to adapt to support additional product differentiation via customer-specific value-added services, and expand product capability to new features or standards via a common hardware platform. This will allow for quicker system design and validation and the reuse of existing application code. It will also result in a much more scalable architecture that can easily take advantage of advances in CPU and packet-processing capability from one silicon generation to the next.

The extended programmable processing represented can also enable an evolution towards a network and computing infrastructure that is fully autonomic i.e., one that can diagnose its own problems and respond with little to no operator intervention; one that automatically senses shifts in demand (or status of available resources) and adjusts its behavior to provide the optimal response; one that identifies critical resources and accelerates access to them; and one that is easy to install, provision, repair, upgrade, and afford. Autonomic features could include the following:

- Load-driven provisioning and balancing of services.
- Self-healing through automatic discovery and alerting of faults.
- Intrusion prevention whereby instead of just simply logging the fact that an attack has occurred, the enterprise is protected from the malicious entity.
- Protected (authenticated) management of the platform.

Expense Reduction with the Converged Edge

The focus of IT managers on reducing operating expenses is supported with the inherent value of converging multiple Enterprise Edge functions into a single platform. This hardware integration, in conjunction with continued advances in operating system and management tools, will deliver a number of significant benefits. As outlined in the introduction, this

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

converged solution will support reduced capital and operating expenses by having fewer numbers and types of unique boxes deployed, by requiring less IT operations support due to the reduced box count, and by a migration to a unified network and computer management system.

THE BACK-EDGE: A DRAMATIC SHIFT TO VIRTUALIZED, NETWORKED STORAGE

Over the last several years, the storage market has been undergoing a significant and rapid transition away from a Direct Attached Storage (DAS) model (storage installed inside a particular server) to a Networked Storage model. Networked Storage includes both Fibre Channel-based Storage Area Networks (SANs) and Ethernet-based Network Attached Storage (NAS) appliances (see Figure 3).

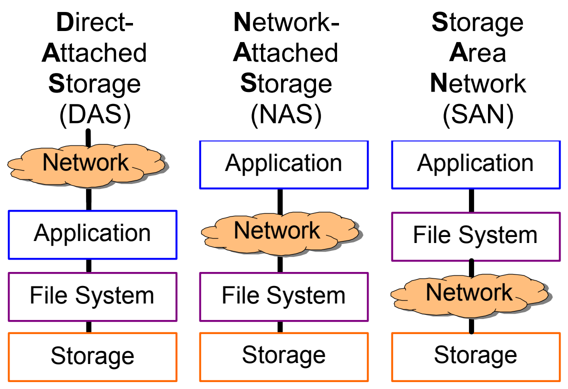


Figure 3: Simple view of DAS, NAS and SAN

The shift to networked storage has opened the possibility to view storage as an independent resource that can be managed and optimized more efficiently than can be done with DAS. The economics underlying the transition to networked storage are very compelling. When viewed from an IT manager's perspective, the impact of a SAN implementation (vs. a DAS implementation) typically includes the following:

- 75% more storage capacity (GBytes) handled per IT manager.
- A 70% increase in disk utilization.
- A 50% reduction in costs for back-up hardware.
- Less data center floor space (due to increased density).
- Fewer general-purpose file servers.

While most network storage implementations today are Fibre Channel-based SANs, similar benefits can be accrued by LAN (Ethernet)-attached NAS boxes. The costs associated with storage parallel those seen in the server world, with ongoing maintenance, backup and provisioning making up the majority of the total cost of ownership.

Network Storage Today

The transition from DAS to NAS has been underway for some time, though it is far from complete. The storage infrastructure will typically include the elements in Figure 4

.

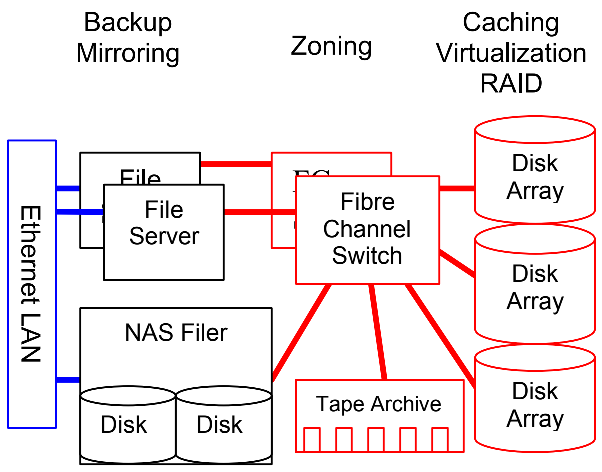


Figure 4: Today's back-edge storage configuration

Fibre Channel Storage Area Networks

The SAN infrastructure is a totally separate network with all of the normal costs and complexities involved in any large network. A series of Fibre Channel switches provide the interconnect within this network, while the connections to the LAN are through either a dedicated file/database server or via a NAS appliance.

In this model, the actual Fibre Channel SAN is relatively "dumb," providing network connectivity and some very basic services (zoning, etc.). The switches are roughly equivalent in functionality and complexity to managed Layer 2 Ethernet LAN switches that support Virtual LANS (VLANs). Fibre Channel directors (large, modular switching platforms) are found in large installations. These products provide larger port counts and greater reliability via redundancy, etc., but are still essentially "dumb" switches.

Fibre Channel SAN implementations span a range of performance levels with 1 Gb/sec and 2 Gb/sec links representing the majority of the installed base of switch ports. During 2003, the market has begun to transition to

4 Gb/s switch ports. This progression is expected to continue to include 10 Gb/sec Fibre Channel links in the 2005-2007 timeframe. 10 Gb/sec links will initially be relegated to aggregating traffic from multiple switches/directors or for switch to switch links, though native 10 Gb/sec server and array connections will begin to be deployed during this period.

Disk Arrays and Storage Subsystems

The actual data storage elements reside in the end-points of the Fibre Channel network: disk arrays and disk subsystems. The basic architecture of a storage subsystem includes one or more channel controllers (the Fibre Channel network interface), a number of disk controllers (providing the interface to a series of physical disks, typically via a Fibre Channel-arbitrated loop) and an array controller (a processor complex that manages all of the functions of the array and provides storage services).

The array and subsystem platforms provide many of the intelligent services delivered in the SAN including the following:

- *Redundant Array of Inexpensive Disks (RAID).* To provide improved data integrity and immunity from disk failures.
- *Disk caching.* To provide improved response time for frequently accessed data.
- *Disk virtualization.* To allow "clients" to reference a logical disk where the actual data storage may exist anywhere on one or more physical disks, thus improving utilization.

The services provided by the disk subsystems represent only a fraction of the services demanded of an enterprise storage network. Many additional services must be supplied by the computing elements in the network (the general-purpose server platforms acting as file servers). File servers typically provide the following services:

- *File system services.* These implement the actual file system and maintain the mapping of files to actual blocks of data on a disk. Typical file system support would include CIFS, NFS, NTFS, etc.
- *SCSI processing.* This implements the SCSI protocol for accessing block data from the disks/subsystems.
- *Backup, mirroring, etc.* The file servers are often responsible for directly managing the data movement required to mirror databases and perform backups for both archiving data and remote backup that provides for disaster recovery.
- *Storage management and monitoring services.*

Network Storage Tomorrow

The back-edge of tomorrow will be based on platforms that deliver the same services deployed today in a much more consolidated, easy to manage system. This will include the ability to consolidate around a single data center network, seamlessly support both **block and file services** (using multiple file systems) with the same infrastructure and the ability to integrate cleanly with legacy file servers, Fibre Channel switches, and storage subsystems. These platforms will also serve as the deployment point for a range of services that are either totally new or are deployed at a different point in the network.

Network Attached Storage

NAS products (often referred to as "filers") represent a step in the evolution toward a more consolidated back-edge of the enterprise data center. They provide the means to deliver many of the benefits of SAN-attached network storage without the necessity to install and maintain a separate Fibre Channel network.

The initial NAS products were storage appliances (often based on a standard PC/server platform) that could be attached to a standard LAN and act as a dedicated file server. NAS products represented a low-cost way to deploy networked storage, and they took advantage of the file system code running on the PC platform.

NAS appliances could include direct attached storage or could act as a front-end to a Fibre Channel SAN (referred to as a NAS head). In either case, the primary function of the NAS box is to implement the file system, taking in file requests and generating the appropriate SCSI commands to read/write the actual disk blocks. As the popularity of NAS products has grown, the trend has been to develop more hardware platforms specifically designed for NAS applications, thereby improving cost, performance, and scalability. The back-edge of tomorrow will extend the NAS concept to support multiple protocols and offer multiple services.

The initial deployment of NAS products in the enterprise provided incremental storage for a department or workgroup. The current usage model now includes NAS as a critical part of the total enterprise storage infrastructure. The overall trend is toward the convergence of both NAS and SAN deployments into a single, unified storage infrastructure.

Multi-Service Storage Platforms

The Multi-Service Storage Platform (MSSP) is a product concept for the next generation of storage infrastructure. The MSSP (see Figure 5) will allow an enterprise to extend the data center LAN to deploy an all IP-based storage network that offers the performance, reliability, and scalability that has come to be expected in a dedicated Fibre Channel SAN. The MSSP will be compatible with the legacy SAN infrastructure, allowing the IT manager to migrate to the IP infrastructure at a pace that makes sense for the business. The MSSP will also allow the deployment of multiple storage services at one central point in the network, thus reducing management complexity and expense and expanding the scope of a given service to include the entire infrastructure.

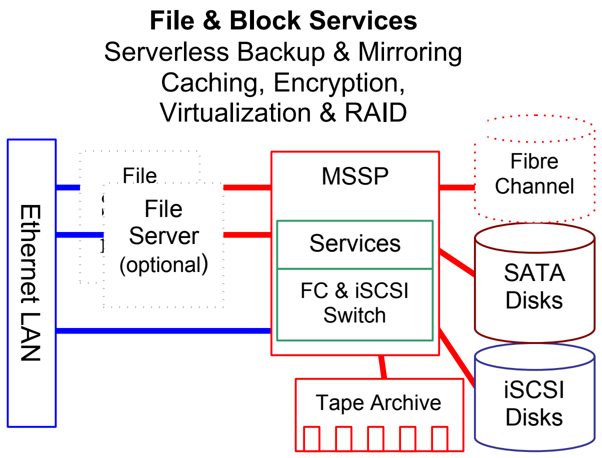


Figure 5: Tomorrow's Multi-Service Storage Platform

The MSSP supports the deployment of a number of new or improved protocols and services in the storage infrastructure. Some of the key services and protocols are discussed below.

iSCSI

Storage networks have traditionally been physically separate networks that provide logically different services: Ethernet LANs have hosted file-based storage traffic and Fibre Channel SANs have hosted block-based SCSI storage traffic. The iSCSI (IP SCSI) protocol allows the convergence of these logically and physically disparate networks. iSCSI provides the means of delivering SCSI commands (typically carried via Fibre Channel in a SAN) via an IP network.

The IP network may be an Ethernet LAN or any IP-based Wide Area Network/Metropolitan Area Network (WAN/MAN). iSCSI is seen as a way of extending the reach of SANs across large geographic distances (beyond the reach of Fibre Channel) to support remote storage.

iSCSI deployments are expected to be minimal over 1 Gb/s Ethernet (suffering from lower performance relative to 2 Gb/sec and 4 Gb/sec Fibre Channel links). As 10 Gb/sec Ethernet links are rolled out in the data center, iSCSI will likely begin to be viewed as a mainstream alternative to Fibre Channel.

The MSSP will support iSCSI at multi-Gigabit rates as a means of delivering block-based storage access via the IP network. This is key to enabling a converged LAN/SAN environment.

Virtualization

Virtualization of storage resources has been proven to have a strong positive impact on provisioning, availability, utilization, and maintenance (operating) expenses associated with storage in the data center.

Virtualization encompasses a number of different technologies, applying to both block- and file-level virtualization. In either case, the concept is to decouple the logical representation of a piece of data (either block or file) from the physical instantiation of the data. This requires a mapping of the logical reference to one or many different physical structures (on the same or different storage subsystems).

The MSSP will support both block and file-level virtualization services at a central point in the network. This will allow the benefits of virtualization, transparently to the application, to span multiple hardware platforms from multiple vendors. In addition, management of the virtualization services will be focused on a single, central platform rather than across multiple disparate vendor platforms.

Disk Caching

Disk caching is a very powerful tool for improving the performance of storage subsystems. Caching is often closely coupled to the virtualization system (since it is the logical elements that an application cares about and will want to cache). Disk caching requires sophisticated algorithms to predict the data most often accessed and to store it in a local DRAM so it can be accessed without incurring the latency of reading from a physical disk. Disk caches also function as write buffers so that applications can "write and forget" without incurring disk-write latencies. Data integrity is a primary concern for disk caches. Vendors go to great lengths to ensure that data in flight (cached or a buffered write) is immune to bit errors.

Both virtualization and disk caching are very compute-intensive applications that deal with very large active data sets (a disk cache may use 16-32 GBytes of DRAM for its cache).

Both caching and virtualization services in the MSSP will require very high compute performance and the ability to manage very large databases. For this reason, a large percentage of these applications are hosted on Intel Architecture microprocessors.

Secure Storage

Security technology is not widely deployed within storage networks. Historically, Fibre Channel SANs have benefited from their physical separation from the bulk of the enterprise (accessible only via dedicated file servers with access control capabilities). With the introduction of LAN-based technologies, such as NAS and iSCSI, the isolation of the Fibre Channel SAN is lost, particularly when IP networks are extended outside the enterprise. Recognition of this has led to the requirement for IPSec (the IP Security protocol) as part of the iSCSI protocol.

In addition to providing security for data in transit (e.g., when using iSCSI for Internet-based remote backup), a number of new environmental factors are driving a move to secure data at rest. The primary drivers for this move are based on an increasing need to protect private information–including legislative requirements.

The Health Information Portability and Accountability Act (HIPAA) prescribes that health care providers and insurers take steps to maintain confidentiality of patient records and that they retain records for extended periods. Many have interpreted this act to mean that medical records stored on disk or archived in electronic format must be encrypted to ensure confidentiality.

In addition to health care requirements, new standards of corporate accountability require that information regarding financial performance and reporting not only be retained but that the integrity of the data (freedom from tampering) be certifiable. This is most readily achieved by using encryption and authentication techniques (such as those used in IPSec).

In order to provide the level of security demanded, and not interfere with the expected performance and availability of the data storage encryption/decryption, processing must be handled on the fly (at full line rate). Further, the crypto processing must be compatible with the other required services, such as caching, virtualization, backup. This requires very high-performance packet processing to complement the applications processing.

Application Servers: High-Performance Host Media Processing

Much of the focus of this paper has been on applications with a very high network bandwidth component. These offer clear examples of cases where the application of high-performance packet processing has a direct impact on the overall system performance. There are also applications, however, where the network bandwidth requirements are low that can still benefit from a coupling of the capabilities of the IXA and IA product families. Host-based Voice Band Media Processing (HMP) is such an application.

An IP media server is a solution using host media processing running on high-performance computing platforms using the Internet Protocol (IP) to provide media services to the enterprise telephony infrastructure. These services include voice, fax, speech, and audio conferencing services to augment and in the future replace PBX solutions, as a back-edge solution.

An IP media server could be used to deploy an Interactive Voice Response (IVR) system. IVR works like this. A media server answers a call, plays and streams a voice file, and detects digits (via a DTMF mechanism) for interactive selections. IVR is currently being deployed in such applications as financial services, (telebanking) and airline travel reservation call center call flow management.

Today’s IP media servers operate on standard server platforms without acceleration hardware, supporting a maximum of ~400 IVR channels. Tomorrow’s high-performance enterprise applications require 480 IVR channels, while telecommunication applications will require a minimum of 672 channel density. One approach to achieving these rates in tomorrow’s IP media server is to increase supported channel density by offloading packet processing from the IA32 server to a network processor.

The technical challenge is in handling G.711 traffic with 10 msec frames, short RTP packets (80 byte payloads with 54 bytes of overhead) and 1 frame/packet. Each IP IVR channel has 200 packets per second to be processed (100 each direction). While the aggregate data rate is relatively low (<55 Mb/sec total for 500 channels), the overhead of managing a constant stream of small packets taxes even very high-performance server CPUs.

A number of scenarios were analyzed to determine the optimal partitioning of tasks between the IA processor and the network processor. The scenarios ranged from no packet processing offload using standard 100BASE-T and 1 GigE NICs through solutions with RTP (real-time transport protocol) and IP media offload by using IXP technology on PCI as well as a solution similar to the TWIN Cities Experimental prototype where the IXP and IA communicate through a shared DRAM architecture (see Appendix A).

The real benefit of the IP Media offload is reducing the IA32 media processing footprint (MHz) per channel, permitting more channels to be serviced per IA32 CPU. The improved compute density can also result in reducing the solution's physical size and power requirements. The number of IP IVR channels supported per server will vary depending on the type of offload solution. A shared memory inter-processor communication mechanism, similar to that used in the Twin Cities Experimental Prototype, is expected to yield the best results.

For an application of an IP media server, a typical IA server platform would be used, which includes the following:

- Dual 3.2 GHz Xeon CPU-based platform, with hyper threading (4-way simultaneous multithreading)
- 512 MB of SDRAM (minimum)
- A suitable chip set such as Intel's E7501

With this system configuration, the number of supported IVR channels will range from a low of approximately 400 with no packet processing offload to an estimate of over 700 using a solution similar to the Twin Cities shared memory approach.

The greatest gains at the system level can be realized by tightly coupling the packet processor with the IA compute engine, thereby minimizing the amount of overhead necessary to communicate between the two systems.

In summary, when tomorrow's high-performance IP media server combines edge router functions with media processing, the benefits to both the IXP family of network processors and IA promises to impact system performance even when the actual network bandwidth needed is less than 100 Mb/sec.

The Common Threads–Silicon Requirements at the Converged Edge

When analyzing opportunities for convergence at the front and back-edge of the enterprise data center, a number of common themes become apparent.

The most obvious is that there is a strong and growing need for improved packet-processing capabilities in the data center. The demands placed on edge equipment at Gigabit Ethernet rates and beyond require specifically targeted packet-processing silicon solutions.

A second important theme is that the demand for high-performance application processors is virtually insatiable. This is particularly true in applications where content processing is required on entire network data streams. It is also clear that the mix of packet processing and application processing will vary across different applications–aligning well with a scalable building-block solution.

A more subtle observation is the opportunity for improving the performance and time-to-market of edge systems by more tightly coupling the packet and application-processing silicon.

Intel's current portfolio includes a range of products targeted at addressing the needs of the converged enterprise edge. A brief summary of some of the characteristics of the relevant products follows.

EDGE PLATFORM BUILDING BLOCKS

Intel Corporation produces both a high-performance application processor, the Intel Xeon processor and a high-performance packet processor, the IXP2850 network processor. These processors are well suited for edge applications. When used together they deliver formidable edge performance across the range of edge applications described in the previous sections.

This section describes the two processors and their suitability for various edge application tasks. Then, we describe platform-level communication between the two processors. Finally, possible future enhancements to this communication are explored.

Intel Xeon Processor in the Edge

Intel Xeon processors are commonly found in front-edge devices such as intrusion-detection systems, SSL accelerators, traffic managers, Virtual Private Network (VPN) gateways and in back-edge devices focused on providing high-performance storage. The Intel Xeon processor is compelling for these applications because it offers both a high-performance/low-cost platform and a high-productivity development environment.

The Intel Xeon processor features a high-performance IA core with a large cache (as of 10/7/03 a 3.2 GHz core with a 1 MB cache[1]). The platform surrounding the processor features low latency access to high-bandwidth memory (up to 4.3 GB/sec). Dual-processor configurations are available and prevalent in the edge. In these configurations, two Intel Xeon processors are presented with a shared, coherent view of main memory, enabling easy and rapid sharing of packet data and state. The platform allows for a high-throughput network connection through Network Interface Cards (NICs) placed in its Peripheral Component Interconnect (PCI) or PCI-X busses.

Edge application code can be developed with standard IA development tools such as the GNU C compiler (GCC)[2] and Visual C++[3] already familiar to developers. Moreover, developers can often utilize open source code like Linux Netfilter[4] Firewalls and OpenSSL[5] to speed code development.

Platforms based on the Intel Xeon processor excel at complex processing over the large state required by edge applications. Traffic management is a good example. For these applications, IP packets must be combined into Layer 7 requests. Next, packet data are searched for strings that identify the server that can best service the request. For example, cookies within HTTP *Get* requests identify the server that has already performed a public key exchange with a client and so can most efficiently service the request. Intrusion Detection is more complex, requiring not only searching within a given packet stream, but also searching through state held on all flows to identify attempted multi-flow attacks on a site (such as port scanning). Systems based on the Intel Xeon processor offer an excellent platform on which to write, debug, and execute the large body of code needed to run these applications quickly.

[1] "Intel® Xeon™ Processor." Intel Corporation. http://www.intel.com/products/server/processors/server/xeon/index.htm?iid=Homepage+STT_xeonproc_03ww41b&

[2] "Welcome to the GCC Home Page." http://gcc.gnu.org/

[3] "Microsoft Visual C++*." Microsoft Corporation. http://msdn.microsoft.com/visualc/

[4] "Netfilter, firewalling, NAT and Packet Mangling for Linux 2.4." http://www.netfilter.org/

[5] "Welcome to the OpenSSL Project." http://www.openssl.org/.

IXP Network Processors in the Edge

IXP network processors are architected specifically for the task of packet processing. Traditionally, packet processing at high line rates has been done by special-purpose Application Specific Integrated Circuits (ASICs). The drawbacks to ASICs are well known: high development costs, long development cycles, and lack of flexibility (e.g., they are designed for a specific, single function and can not evolve to meet the need for new functions and features). There are two specific problems associated with packet processing on a general-purpose microprocessor.

Packets can arrive at a faster rate than a general-purpose processor can handle. For example, for Gigabit Ethernet, the frame arrival rate can be as many as 1.5 million frames per second (one frame every 672 nsec). For a 2 GHz processor, that equates to 1300 CPU instructions. For some edge applications this may not be enough for the required processing and packet IO. Also, cycles spent here subtract from the time available for application processing.

To compound the problem, the general-purpose processor often doesn't get to use all of the cycles for productive work. Memory latency causes lost processing time. Processing packets requires accessing and updating state stored in memory, for example information defining a given TCP connection to which a packet belongs. Main memory has a long latency in terms of processor cycles (for example a 100 nsec DRAM read is equivalent to 200 cycles on a 2 GHz processor). The large number of flows in many edge applications drives an increase in cache miss rates as the cumulative state and packet data for all the active flows becomes too large for the CPU's cache. Moreover, a large number of small packets increase the miss rate by requiring many small network IO operations, from the CPU to the NIC or to main memory. Each cache miss results in many wasted CPU cycles. Increasing CPU clock rates exacerbate this problem.

Network processors are designed specifically to minimize the performance impact of these problems using the following techniques:

- *Multiple RISC processors*. Intel's IXP processors contain specialized RISC processors called microengines, which are tailored specifically for packet processing. microengines are designed for parallel processing via multiple instantiations in arrays. They have an instruction set optimized for packet-oriented processing and are much smaller in silicon area than a general-purpose microprocessor. For example, Intel's IXP2800 contains 16 such microengines, running at 1.4 GHz for a total peak processing capability of 22.4 billion instructions per second.

- *Multiple threads per processor*. The issue of long memory latency applies to network processors just as it does to general-purpose microprocessors. One approach to mitigate the impact of memory latency is to process many packets in parallel, using multiple threads per processor (Intel's IXP microengine contains 8 threads). Each thread handles a different packet. While one or more threads waits for the return of a memory read, another thread processes its packet. Specialized hardware in the network processor enables the parallel threads to maintain packet ordering and synchronization.
- *Multiple memory types.* During packet processing, two types of information are accessed, packet payload information, and control information. Packet payload tends to be accessed in large chunks (e.g., 64 bytes) which maps well to the IXPs directly connected DRAM. Control information often consists of small data structures (e.g., 4 or 8 bytes). Accessing a 32 byte cache line to modify 4 bytes results in very poor memory bandwidth utilization. Therefore, the IXP network processors also connect directly to SRAM, which performs well for small accesses.

As explained earlier, edge applications include significant amounts of packet processing. IP routing and Network Address Translation (NAT) are at the core of most packet-processing applications. Edge applications are no exception. The architecture of the IXP allows it to efficiently route and NAT packets, even streams of exclusively minimum-sized packets. The IXP2850 includes hardware acceleration for bulk encryption/decryption required by IPsec and SSL VPNs. The IXP2800 is also capable of terminating TCP connections over 100s of thousands of flows at multi-gigabit rates with minimum-sized packets.

Network processors are designed specifically for the lower-level packet processing described here, and generally not for higher-level application processing. General-purpose CPUs also benefit more directly from 30 years of compiler, debugger, language, and algorithm development and use. While software development tools, such as compilers, are available, a network processor's hardware may not be as fully abstracted from the programmer as current general-purpose CPU hardware. Network processors, for example, may place limits on program size and may require additional low-level coding. Network processors enable high-performance IO and hide memory latency delays, but also require some additional programmer management of resources.

The strengths of the IXP complement the strengths of the Intel Xeon processor. A platform that includes both the IXP processor for packet movement/packet processing and the Intel Xeon processor for higher-level packet processing/application processing promises to deliver leading performance for edge applications.

IA/IXP Communication–This Generation

The converged applications described in this paper require the NP and CPU to share payload data and state information. In today's generation of IXP and Intel Xeon processors, information is transferred via the PCI bus. Each processor has its own private memory space. Buffers for passing packet data are set aside in each of the memories. A packet is passed by first allocating a buffer, writing the packet data into the buffer, and then posting a pointer onto a message queue.

The message queue is often managed using a ring in memory. Rings (also known as circular queues) are often used to pass data or messages from a producer of packet data to a consumer. The ring provides for rate matching enabling the long-term matching of production and consumption rates over short-term mismatches. The ring itself is a block of memory that is accessible to both producer and consumer. (Possibly by a window into memory through the PCI bus.)

The producer maintains a tail pointer, which indicates where the next message should be put onto the ring. The producer writes the message into memory at the address indicated in the tail pointer and then increments the tail pointer. The consumer maintains a head pointer, which indicates where the next message should be removed from the ring. The consumer reads a message from memory at the address indicated in the head pointer, and then increments the head pointer. The producer and consumer periodically read the head and tail pointers (respectively) to ensure that the ring does not overflow or underflow.

The two processors need to pass not only packets but also state. State is generally held per flow and can be quite large for edge applications. Per flow state sizes of 1,000 bytes or greater are not uncommon. Since average HTTP *Get* requests are about 400 bytes in length, state traffic may easily overwhelm packet traffic. Current edge platforms can use the same circular queue described above for passing state. The following options are available:

- The required state can be passed along with the packet. Here the network processor is the owner of the state. State is appended to packets as it is passed to the CPU. The NP must make sure that all the required state is included and labeled. The CPU must then interpret state in the message as necessary and pass all state modification messages back to the NP. The NP must decode and act upon these messages, updating its local state. Multiple copies of some state variables may be required to correctly handle multiple packets on the same flow.

- The NP and the CPU may each hold local copies of state. Here each processor may consult its own local state copy when processing a packet. State updates require both writes to memory to update local state and the creation and passing of a message to the other processor. The receiving processor decodes the message and performs the state update. Careful design is required to ensure the two state copies remain coherent.

Using the example methods described in this section, the Intel Xeon processor and the IXP processor can collaborate to deliver excellent edge application performance and features, better than either processor could deliver on its own.

IA/IXP Communication–Next Generation

As discussed above, efficient state sharing is essential to achieving high performance for edge applications. The state sharing schemes outlined above make use of the strong write performance of current NP to CPU connections. They explicitly minimize the number of reads required.

Current state-sharing techniques have a number of drawbacks. First they require processors to construct messages around the state to be shared. These messages identify the state and the operation required on the distant state. Constructing and sending these messages incurs both processor instruction and bus cycle overhead. Similar overheads are imposed on the receiving processor to unpack and process the state message. Current state-sharing techniques also add code complexity to edge applications. Developers are required to write and debug code to implement messaging schemes that ensure coherency between multiple copies of state.

Future generations of edge platforms could make sharing of state between the IXP and the Intel Xeon processor more efficient by providing shared memory like that provided on the dual Xeon processor platform. A portion of memory would be shared with coherency enforced by hardware. Both processors would be provided high-bandwidth and low-latency reads and writes to this portion of memory. Moreover, each processor would be able to perform atomic read-modify-write operations on this memory. Such a shared memory would allow NP and CPU programs to simply read and write shared state, removing the platform performance overheads and programming overheads inherent in state sharing on current edge platforms. This shared state has been prototyped on Twin Cities (see Appendix A) where strong application-level performance advantages were measured.

Shared state would also enable more natural migration of functionality between the two processors. It is likely that new edge functionality will be placed first on the Intel Xeon processor. As this functionality matures and becomes important to a larger fraction of packets, it may be migrated to the IXP. If the process implementing the functionality communicates with other application processes through shared memory, then it may be migrated without altering those other processes. Processing is migrated from one processor to the other, but state and packet handling interfaces remain unchanged in the shared memory. Such a programming model promises to deliver time-to-market advantages for the introduction and acceleration of new edge functionality.

CONCLUSION

The enterprise edge is getting smarter and getting simpler. Intelligence deployed in the network is seen to deliver a number of important benefits, including improved security, better end-to-end (user-to-user) performance and reduced management complexity. Greater intelligence allows for the deployment of fewer, more converged systems at the edge.

There are tremendous benefits to be realized as a result of convergence at the edges of the enterprise data center. Convergence at the platform level is seen as a way to reduce capital and operating expenses by reducing the number and type of platforms that must be deployed and managed. In addition, these converged platforms are being designed to deliver a range of new services that will offer improved security, better reliability and manageability, and the flexibility to support new services via field upgrades.

Delivering these converged platforms will require a paradigm shift in the platform architecture and the underlying silicon. The ability to consume and process multi-gigabit network data streams while delivering rich application performance is beyond the capabilities of today's general-purpose CPU architectures alone. Intel has recognized this necessity and has developed a family of network processors optimized for managing the high-performance network streams. The network processor family is designed to complement the application-processing capabilities of the embedded Intel Architecture CPUs.

Edge platforms that exploit the IXP and IA processor families offer the high-performance packet and application processing demanded by today's and tomorrow's converged edge platforms.

As the services deployed become more complex and the bandwidth of the network increases, more work is needed to keep pace. The converged platforms at the edge will increasingly require very tight coupling between the applications and packet processors. As more of an application is offloaded to a multi-processor or multi-threaded architecture to address throughput and latency, the degree of interaction (typically sharing of state) between the two domains will grow.

Intel's research has identified a number of architectural approaches that can be employed to allow the accelerated packet and application processing described above, while maintaining the ease of programming and the wealth of available tools and applications available for the Intel Architecture processors.

A system that combines the IA and IXP products represents the fusion of two different architectural approaches. Harnessing the best attributes of each in a single system helps deliver new levels of performance for existing applications while enabling new classes of application that were not previously possible.

Appendix A

Twin Cities: An Experimental IA and IXP Convergence Prototype

The Twin Cities prototype[6] was constructed to investigate the viability and impact of a higher performance platform coupling between the Intel® Pentium® processor and the IXP. The prototype demonstrates that these two processors can be attached with a high-throughput, low-latency, cache-coherent bus and that such a connection can deliver higher application performance.

Twin Cities connects the IXP1240's SRAM and DRAM busses to the Pentium® III processor's CPU bus as shown in Figure A-1. A dual port SRAM (DPSRAM) connects directly to the IXP1240. The other side of the dual port SRAM connects to the CPU bus (a.k.a. Front Side Bus or FSB) through translation logic implemented in the Twin Cities Field-Programmable Gate Array (FPGA). Twin Cities employs a *copy* architecture. Writes of data to be shared between the processors go to both the IXP1240's SRAM (or DRAM) and the Pentium III processor's DRAM. The dual writes don't waste instruction cycles on either processor, since they are hardware managed. Reads of shared data always come from the processor's local memory.

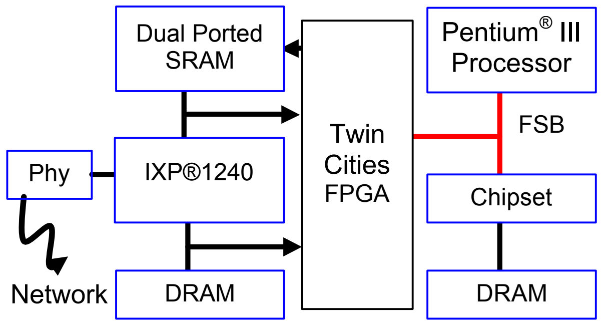


Figure A-1: Twin Cities architecture

Figure A-2 is a picture of the Twin Cities system. The Twin Cities board plugs into the IXP board's SRAM and DRAM connectors through 5 Miktor connectors and into the motherboard through a Slot 1 card edge connector. The compact PCI backplane only provides power to the IXP card; the PCI protocol signals are left unused.

Portions of the Intel® IXA SDK 2.0[7] Resource Manager were ported to Twin Cities to enable structured communication between the two processors and to enable reuse of existing IXA code. The IXA SDK 2.0 Network Address Translation (NAT) application runs on top of this infrastructure. For NAT, the microengines independently perform packet modification and movement for known flows. The core of a virus-scanning firewall application runs on top of NAT. Packets on identified flows are forwarded to the IA processor, which reassembles these packets, and searches them for 32 different virus strings using a Boyer-Moore

[6] F. Hady, T. Bock, M. Cabot, J. Chu, J. Meinecke, K. Oliver, W. Talarek. "Platform Level Support for High Throughput Edge Applications: The Twin Cities Prototype." *IEEE Network*, July/August 2003, Vol. 17, No. 4.

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

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

[7] Intel Corporation, "IXA Software Developers Kit 2.01 for IXP1200," http://www.intel.com/design/network/products/npfamily/sdk2.htm

string search algorithm. Packets containing viruses are dropped by the IXP, while clean packets are forwarded.

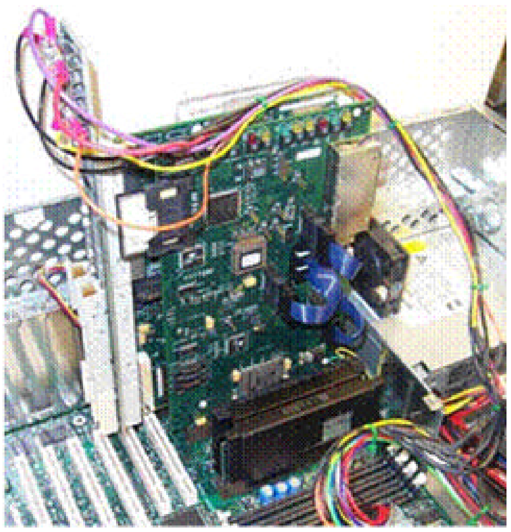


Figure A-2: Twin Cities prototype

Our low-level performance characterizations show that Twin Cities achieves four times the throughput and a quarter of the latency of an IO bus (PCI) connected platform. We measured CPU to NP and NP to CPU write performance of 266 MB/sec on Twin Cities. Karlin[8] measured transfer rates for very similar IXP1200 and Pentium III processor systems connected over a PCI and found a maximum CPU to NP write throughput of 69 MB/sec, a 4x Twin Cities advantage. Latency was measured by passing control, implemented as a shared semaphore, from an IXP microengine to the Pentium III processor and back. Twin Cities has a round-trip synchronization time of 1.4 microseconds. This same measurement on a PCI connected platform shows a latency of 5.5 microseconds, 4x the latency of Twin Cities.

The low-level performance differences measured translate into application-level performance differences. NAT packet forwarding for already established connections is handled by the IXP1240's microengines, so throughputs didn't differ between Twin Cities and PCI connected platforms. Packets sent on new connections, however, require the packet to be sent to the Pentium III processor, which then sets up an entry in the NAT table.

We measured 45,000 new connections per second for Twin Cities and 23,000 new connections per second for our PCI-connected platform. For the virus scanning firewall, all packets must be sent to the IA processor. Control traffic is sent in both directions. For this application, Twin Cities was able to maintain a throughput of 200 Mbits/sec while the PCI connected platform achieved 45 Mbits/sec.

[8] S. Karlin, L. Peterson, "VERA: An Extensible Router Architecture," *Computer Networks,* Volume 38, Issue 3, 2002.

ACKNOWLEDGMENTS

There are a few key individuals who have provided tremendous support and path finding. The authors thank Debra Towle, Myles Wilde, Tony Bock, Duane Galbi, Mason Cabot, and Jon Krueger.

REFERENCES

[1] B. Carlson, "Intel Internet Exchange Architecture and Applications, A Practical Guide to IXP2XXX Network Processors," *Intel Press*, 2003.

[2] Erik J. Johnson and Aaron R. Kunze, "IXP2400/2800 Programming, The Complete MicroEngine Coding Guide" *Intel Press*, 2003.

[3] F. Hady, T. Bock, M. Cabot, J. Chu, J. Meinecke, K. Oliver, W. Talarek, "Platform Level Support for High Throughput Edge Applications: The Twin Cities Prototype," *IEEE Network,* July/August 2003, Vol. 17, No. 4.

[4] Intel Corporation, "IXA Software Developers Kit 2.01 for the IXP1200," http://www.intel.com/design/network/products/npfamily/sdk2.htm

[5] S. Karlin, L. Peterson, "VERA: An Extensible Router Architecture," *Computer Networks,* Volume 38, Issue 3, 2002.

AUTHORS' BIOGRAPHIES

Matthew Adiletta is an Intel Fellow and is responsible for Intel's network processor development. This responsibility includes evangelizing Chip-Level Multi-Processing (CMT) and hardware-based Light Weight Threading. He led the development of the IXP1200, the first fully programmable network processor for Intel. Adiletta graduated with Honors from the University of Connecticut in 1985 (B.S.E.E.) and currently holds 24 patents. His e-mail is matthew.adiletta at intel.com

John Beck is an engineering manager in Intel's Communication Infrastructure Group Technology Office. He joined Intel in 2000 and has managed the development of IXP1200 and IXP2800 families of network processors. Prior to Intel, he was Director of DSP design at Analog Devices Inc. and Engineering Manager in Digital Equipment Corp. Semiconductor Group. John has B.S.E.E. and M.S.E.E. degrees from

Cornell University. His e-mail is john.c.beck at intel.com.

Douglas Carrigan is a strategic marketing manager in Intel's Communications Infrastructure Group Technology Office. He has held this position at Intel for the last five years and has led the marketing of the IXP family of network processors since the family's inception at Digital Semiconductor in 1996. Prior to joining Intel, Doug was responsible for marketing of DEC's StrongARM processors into embedded applications and led strategic marketing of MIP processors at IDT. Doug has over 20 years of experience in design, sales, and marketing of embedded microprocessors into communications applications. His e-mail is douglas.carrigan at intel.com.

Mark Rosenbluth is an architect in the Network Processor Division. He has been at Intel for five years and prior to that worked at Digital Equipment Corporation, where he was an architect for PCI Bridges. He also worked on VAX and Alpha microprocessors. He holds a B.S.E.E. degree from Rutgers University. His e-mail is mark.rosenbluth at intel.com.

Bill Tiso is the engineering manager at Intel's Network Building Block Division (NBD) responsible for Host Media Processing Technology planning and development in the NBD CTO Office. Mr. Tiso joined Dialogic Corporation in 1993 and managed the development of many products released as Dialogic and Gammalink. He holds a B.S.E.E degree from Florida Institute of Technology and a M.S.E.E. degree from Polytechnic Institute of New York. His e-mail is bill.tiso at intel.com.

Frank Hady is a principal engineer at Intel leading a small research group focused on providing high-performance network edge platforms. He also serves as the NPF benchmarking Task Group chair. Frank holds a Ph.D. degree from the University of Maryland. He has three patents (with nine pending) and has authored articles on networking platforms, network processor benchmarking, PCI usage, network interface design, and MPP network design and performance. His e-mail is frank.hady at intel.com.

Scalable Intel® IXA and its Building Blocks for Networking Platforms

Bapi Vinnakota, Intel Communications Group, Intel Corporation
Paul Dormitzer, Intel Communications Group, Intel Corporation
Mark Rosenbluth, Intel Communications Group, Intel Corporation
Sridhar Lakshmanamurthy, Intel Communications Group, Intel Corporation

Index words: IXA, wireless infrastructure, wireline infrastructure, edge routers, core metro, IXP network processor, IXF devices, Multi-Service Switches, BSC, RNC

ABSTRACT

Platforms at the edge of the network, such as Multi-Service Switches (MSS) and edge routers, convert traffic from legacy networks to Internet Protocol (IP) traffic for transport across the network core. Line and service cards in these platforms will have to handle traffic in a wide range of protocols and line rates. Current silicon solutions for edge applications are limited in functionality and performance. These limitations increase both the installation and operational cost of edge systems. Scalable Intel® Internet Exchange Architecture (Intel® IXA) consisting of scalable IXP network processors (NPUs) and IXF connectivity devices address the limitations of current silicon solutions. We review the application and benefits of Intel IXA for a wide range of wireless and wireline network edge infrastructure platforms. We also discuss how the internal architecture of IXA components enables the external application scalability. The functional flexibility and scalability of IXA will support the expected convergence of voice and data networks and the delivery of advanced services at these platforms.

INTRODUCTION

A significant amount of processing in the network occurs in platforms at the edge of IP networks such as edge routers and multi-service switches [1]-[4]. As voice and data networks converge, edge platforms will be required to support increasing bandwidth requirements and expand new services while preserving the existing infrastructure. These multi-protocol platforms offer service providers the opportunity to leverage legacy access networks to satisfy the growing need for higher bandwidth and to meet the demand for new voice, video, and data services. Additional capabilities needed from these platforms include reliable aggregation and forwarding, robust Quality of Service (QoS), enhanced reliability, enhanced security and the addition of new protocol standards.

Even as the processing in these platforms is expected to evolve these platforms will have to continue to handle legacy traffic. Traffic from a variety of legacy access networks, Frame Relay/PDH traffic from the enterprise, Asynchronous Transfer Mode (ATM) traffic from broadband networks, POS/Ethernet traffic, is transformed to Internet Protocol (IP) traffic. Traffic from multiple sources is delivered to these platforms in multiple (tens of thousands) logical channels and aggregated before being sent to the network core. Edge routers and multi-service switches are required to support a large number of channelized ports ranging in speed from OC-1/OC-3 to OC-48c/OC-192c. They require support for the unique "any service, any port" paradigm with full flexibility in the building-block components to support any inter-networking protocols on any of the incoming ports. To support a diverse range of access networks, a wide range of functions (protocols and services) and bandwidths has to be realized on line and on service cards in these platforms. The devices on the service and line cards will have to support ATM, Frame Relay, Multi-Protocol Label Switching (MPLS), IP, and TDM inter-networking.

® Intel® Internet Exchange Architecture (Intel® IXA) is a trademark of Intel Corporation or its subsidiaries in the United States and other countries.

Organization

The rest of this paper is organized as follows. First, we discuss the benefits of the scalable Intel Internet Exchange Architecture (Intel IXA) and its components in the context of the limitations of silicon devices currently available in the market. We review the internal architecture of potential next-generation 90nm IXP and IXF devices. We then review the application of IXA to key switching platforms across a range of cost/performance points in the wireline and wireless infrastructure including Multi-Service Switches (MSS)/Edge routers, Radio Network Controller (RNC)/Base-Station Controller (BSC) platforms in 2G/3G wireless networks, and Core/Metro routers. We outline line card architectures and analyze the functionality and performance achieved on the IXA family for these applications.

LIMITATIONS OF CURRENT SILICON SOLUTIONS

Most merchant silicon devices currently available for the range of Layer 1, 2, and 3+ services supported in communication platforms are limited in functionality and performance. Those devices typically perform a small set of fixed functions at a fixed data throughput rate. Layer 3 functions, such as packet forwarding and traffic shaping, are typically realized in custom ASICs. These devices may not enable more than a specific portion of a packet to be examined. At Layer 1 and Layer 2, the functionality is limited to specific subsets of protocols. For example, a device may terminate ATM flows or HDLC flows, not both. Similarly, a device may terminate Plesiochronous Digital Hierarchy (PDH) lines or Synchronous Optical NETwork/Synchronous Digital Hierarchy (SONET/SDH) lines, not both. The maximum bandwidth supported by a device may range from OC-3 through OC-12. The number of logical channels supported is also very limited.

The above-mentioned limitations lead to a number of limitations in line-card design. Current designs for line-cards are protocol- and line rate-specific. A design to terminate a collection of T3 lines differs from one to terminate OC-3 lines channelized down to T3. A design to terminate ATM traffic differs from a design that terminates Frame Relay traffic. A design for an OC-12 card will have more components than a design for an OC-3 card. At higher bandwidths, or when designing multi-function cards, switching silicon will also have to be added to a card to route traffic to the appropriate devices. Most importantly, since the design consists entirely of fixed function components, there is no function flexibility in the line card. As the functionality of the platform evolves, the cards will have to be redesigned.

These limitations have a number of practical consequences. Several factors contribute to the total cost of ownership for both an equipment manufacturer and a carrier.

- High-performance multi-function cards are very expensive. Design and manufacturing costs are very high since the cards are complex and contain a large number of components. Total design costs are also high since a separate card has to be designed for each protocol and line rate.
- Once in the field, the presence of a large number of different designs leads to higher maintenance and tracking costs for both the equipment vendor and the carrier. More components also lead to lower in-field reliability.
- These design limitations also impact carriers deploying edge platforms. Since the cards are line rate- and protocol-specific, the contents of the platform will have to be changed if the traffic mix changes from what was originally anticipated by the carrier.

INTEL IXA

Next-generation Intel IXA offers a comprehensive set of modular building blocks for networking platforms. IXP network processors (NPUs) and IXF connectivity devices, enable the design of a new class of *scalable multi-function, multi-bandwidth line and service cards* for networking platforms. The new class of line cards will consist of an IXP network processor and IXF Phy/Framer/MAC devices. The computational power of the IXP network processor enables flexible Layer 3+ processing and higher-layer services and enables software-based customization. Members of the Intel® IXP 2XXX class of NPUs offer a wide range of performance capabilities. The IXF devices provide the connectivity between the processing components and the transmission medium. IXF components offer flexibility in Layer 1 and Layer 2 processing. Members of the IXF family of devices can process data across a wide range of line rates and multiple Layer 1 and Layer 2 protocols.

The components of the IXA family can be combined in a variety of configurations to meet a wide spectrum of price-performance constraints. For example, at the low end, when terminating less complex protocols, such as Ethernet, Layer 2 and Layer 3 processing can be integrated into one device. When terminating more complex protocols, such as SONET/SDH and/or when a wider range of functionality is required a higher performance solution can be developed using discrete IXP and IXF components.

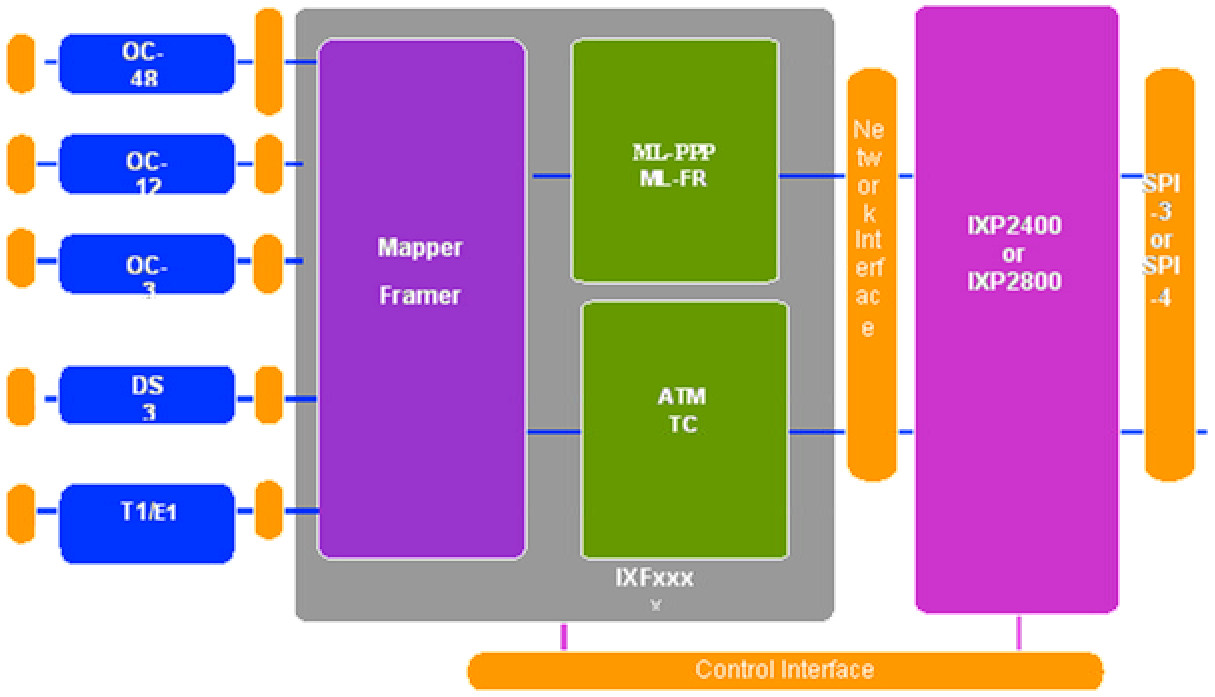


Figure 1: Scalable IXA line card for edge applications

Figure 1 shows one example of how the building blocks of the IXA family, IXF framers, and the IXP network processors can be used together to provide a scalable high-performance SONET/SDH-Layer 3+ line card for edge platforms. The line card design can address all the limitations of current designs. The design can scale across a large bandwidth range and input combinations. The IXF component can terminate SONET/SDH/PDH flows. A diverse set of Layer 2 protocols is supported in the IXF device. The IXP NPUs provide Layer 3+ protocol processing flexibility. The functionality of the card can be altered to meet evolving protocol traffic patterns and service needs. Bandwidth- and function-scalable IXP and IXF components enable a bandwidth- and function-scalable system solution.

SCALABLE NETWORK PROCESSOR ARCHITECTURE

Network processing lends itself to a parallel approach. A network interface can be viewed as an endless stream of packets, each of which can be handled by a separate processor to improve overall throughput. The IXP network processor (NPU) architecture takes advantage of this by providing a large number of specialized processors (and support functions) on a single chip. By so doing the IXP architecture provides more overall processing power than can be achieved in a single general-purpose microprocessor of the same semiconductor process generation. By varying the number and the performance capability of these specialized processors on chip, and by choosing the appropriate physical interfaces, different family members of the IXP architecture are derived. These implementations are targeted to different cost/performance points and provide a single coherent architecture and usage model for a range of network applications. The unique flexibility and scalability of the architecture allows these engines to be programmed efficiently and meet stringent performance requirements for each of the segments.

In the next section, we briefly describe the building blocks of the IXP network processor, and their function.

lists some members of the IXP network processor family, showing how they are built from common building blocks, in differing quantities and capabilities.

IXP NPU BUILDING BLOCKS

The NPU provides the following hardware functions. (The example designs described later in this paper show how these functions are utilized to perform the network processing applications.)

Media Interface

Media Interface refers to an external interface to the network devices: Ethernet, SONET/SDH, ATM, Radio Interfaces, TDM, etc. Different performance points in the market require different performance levels. For example, in SONET, OC-3 and OC-12 are used in the low end, OC-48 is used in the mid range, and OC-192 is used in the high end. For less complex ports such as TDM and Ethernet, the Layer 2 processing can be integrated on chip. For more complex ports such as SONET/SDH, it is done by external components. This provides high integration where needed in the low end and flexibility in the mid and high end. Where external components are used, they are interconnected via industry-standard interconnects. Different interconnects are used in different markets (UTOPIA-2, UTOPIA-3/SPI-3, SPI-4) as appropriate for the price/performance range. However, all IXP NPU family members abstract the interface details as much as possible in order to present a common programming interface.

The Media Interface can also be configured as a Switch Fabric Interface for applications that need to connect to a switch fabric.

Microengines

Microengines are 32-bit RISC processors that do the majority of the network processing tasks such as packet header inspection and modification, classification, routing, metering, etc. The microengine instruction set is a blend of conventional RISC instructions with additional features specifically tailored for network processing. Because the microengines are not general-purpose processors, many tradeoffs are made to minimize their size and power consumption.

Some of their specialized features are outlined below:

- *Multi-threaded*. Each microengine has hardware support for eight threads, with fast thread switching. Each thread has its own private copy of registers, program counter, and other necessary state. Multiple threads allow high utilization on the microengine. Each thread processes a different packet; when one thread waits for long latency operations such as memory reads to complete, other threads can run. This keeps the microengine busy when it would otherwise be idle, if it could only process a single packet at a time. A context switch is done in hardware in a single cycle; no instruction cycles are used to save and restore thread context. In addition, thread switching is completely under software control, allowing the application to be in charge of when threads execute. This allows for coordinated sharing of data.
- *Large register set*. In packet processing, lots of information must be juggled by the application. For example, it may need to read multiple packet headers to parse addresses and other fields, read routing information, update flow state, etc. Having lots of registers per-thread minimizes memory accesses that would be needed to move variables in and out of memory.

Table 1: IXP NPU family members and features

Unit	IXP2400	IXP2800	IXP2850	Possible 90nm NPU
Media Interface	SPI-3	SPI-4	SPI-4	SPI-3/4
MicroEngines	8 at 600 MHz	16 at 1.4 GHz	16 at 1.4 GHz	4-20 at 900 MHz – 1.4 GHz
NPE	N	N	N	1-4
DRAM Interfaces	1 DDR	3 RDRAM	3 RDRAM	1-3 DDR / RDRAM
SRAM Interfaces	2 QDR	4 QDR	4 QDR	1-4 QDR + on-chip SRAM
Scratch	16KB	16KB	16KB	16 KB
Encryption	N	N	Y	Y
PCI (64 bit, 66 MHz)	Y	Y	Y	Y
Integrated MAC and Framer	N	N	N	T1/E1/J1 HDLC, Ethernet
XScale®	600 MHz	700 MHz	700 MHz	1.2 GHz+
L1/L2 Cache	32k/none	32k/none	32k/none	32k/512k

- *General event signaling*. Event signaling is a mechanism that allows threads to sleep while waiting for external events to occur. The events are general in that they can be either notifications from other threads (in the same or other microengines), from other hardware units (for example, notification of data received by the Media Interface), or notification of completion of IO accesses made by the microengine (for example to off-chip memories). Each thread has 15 Event Signals that can be allocated to any of the just mentioned items. The Event Signals are allocated much like registers in the sense that a given Event Signal can be used for different uses at different points in the program. The net effect is that a program is not limited to a total of 15 Event Signals; just 15 (or less) active at any given time. In practice this is rarely a limiting factor.

- *CRC*. Cyclic Redundancy Code (CRC) is commonly used in network protocols to provide protection against errors that can occur in transmission. The microengine provides hardware acceleration for a set of commonly used CRC polynomials, effectively processing 16-bits per cycle (actually 32-bits per instruction, with a repeat rate of one CRC instruction per two cycles).
- *CAM*. The microengine has a Content Addressable Memory (CAM) integrated into its datapath, and a set of instructions that manipulate the CAM. It can be used in a variety of ways to assist in packet processing. One common usage allows threads to efficiently check for data dependencies when they are each handling a different packet. Another usage is to employ the CAM to locally cache connection or flow-related information. Other uses not described here are also possible.
- *Local Memory*. Local Memory (LM) is a block of addressable memory in the microengine, in addition to the conventional general-purpose registers. LM has a wide variety of uses, including caching flow state information, lookup tables, queuing tasks, etc.
- *Multiply*. Some algorithms need to multiply integer values. The microengine multiplier is designed to be low in cost (in terms of silicon area) and flexible in that the time taken to do specific multiplies is proportional to the precision of the data. In other words, it takes less time to multiply an 8bit number by a 16-bit number, than to multiply two 32-bit numbers. This tradeoff provides a good balance between cost and performance.
- *Timestamp*. Timestamp is a 64-bit monotonic counter that can be used as a time reference (it is not a "real-time" clock, however). 64-bits ensure that it will never wrap around.
- *Pseudo-Random Number*. Some network-processing algorithms require a pseudo-random number (PRN). The microengine provides a 32-bit PRN in a low-cost method: each read of the PRN register provides a new random value. Note that this is pseudo-random, not true random as is needed for cryptographic processing.

In addition, microengines are made very area efficient by leaving out the features of a general-purpose microprocessor that are not required for their network-processing task. This allows room for more microengines on chip in a given area. Note the following examples:

- *Memory management*. Because the microengines are running an embedded application, they don't need to run an operating system, and therefore don't need the features provided by a conventional memory management unit.
- *Interrupts*. Microengines provide a software-controlled multi-threading environment, which means that once a thread is running, it will continue to run until it chooses to relinquish control and allow another thread to run. This gives the application complete control. There is no need for interrupts; however, there is a flexible mechanism for interthread signaling.
- *Floating point*. Floating point is typically quite expensive in silicon area, and not needed by network-processing applications.
- *Speculative execution and branch prediction*. Most modern microprocessors with deep pipelines use speculative execution and contain sophisticated branch prediction hardware. In order for branch prediction to work, past execution history must be a reliable predictor of future branch outcomes. This is typically not the case in network processing. The microengine has a relatively short pipeline, and it allows instructions to be executed in branch shadows to minimize cycles that are lost due to taken branches.

Memory

Memory is used to hold network data, as well as support information such as route tables, flow parameters, network statistics, and security associations, among others. Several types of memories are supported, each with different attributes that make it suitable for different uses.

- *Dynamic RAM*. DRAM has high capacity and low cost per bit, which makes it a good choice for buffering large amounts of network data. The low- and mid-range network processors (NPUs) interface to Double Data Rate (DDR) DRAM, which is the same RAM used in mainstream PC products, and therefore has the lowest cost. The high-end NPUs interface to Rambus[*] DRAM. This has a more pin-efficient interface, and therefore allows for more bandwidth,

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

while still keeping the number of pins on the network processor chip within a desired range. The extra feature comes at a small premium in price per bit of memory.

- *Static RAM*: Control information tends to have many small (4-byte, 8-byte, etc.) data structures that need to be independently accessed and modified. Static RAM (SRAM) offers excellent random access to small blocks of data; however, as a consequence, SRAM offers less capacity and costs more per bit than DRAM.
- *Scratch memory*. This memory is similar in function to SRAM except that it is on-chip instead of off-chip. Therefore, it is lower in capacity, but has less latency than SRAM. It is useful for small but frequently accessed data. Scratch memory provides hardware support for Rings (a.k.a. Circular Buffers), which are often used for interthread communication.
- *Cryptographic Processor.* This refers to hardware acceleration of symmetric key cipher algorithms such as DES and AES, and authentication algorithms such as SHA-1. These algorithms are very complex and don't lend themselves to general-purpose computers at high bit rates. Special-purpose hardware to execute these algorithms would be very costly if replicated in each microengine, and it would not be a good balance to the overall system requirements. Therefore the Encryption function is implemented as a shared processor that does the data processing under control of the microengines, which cooperatively share the Encryption Unit.
- *PCI.* This is an industry-standard IO bus. Normally it is used to interface to a system-wide management processor.

Network Processing Engines

Like the microengines, Network Processing Engines (NPEs) are 32-bit RISC processors. While the microengines are optimized for Layer 3 and higher packet and cell manipulation, the NPEs are optimized to synchronize and control a set of coprocessors that each perform a specific data manipulation task, which makes them efficient at Layer 2 processing. They are tightly integrated with their coprocessors which offload compute-intensive processes and bit manipulation and can signal the NPE when they require attention or have completed a task. The coprocessors also have access to the NPE data memory, enabling efficient data sharing between the coprocessor and the NPE core processor. Coprocessors that have been implemented for NPEs include channelized interfaces to High Speed Serial (HSS) ports for T1/E1/J1 framers, Utopia-2 slave ports, HDLC controllers, fast Ethernet MACs, and encryption engines. The NPE core processor's role is to act as a programmable state machine to manage these coprocessors and keep them supplied with data. Because the NPE core acts as a controller rather than a general-purpose computing processor, it can operate at lower clock speeds and power consumption than the microengine and still meet its performance requirements.

The NPE, like the microengine, is hardware multi-threaded. There are 16 hardware contexts, and unlike the microengine they are assigned to three priority levels: higher priority contexts can preempt lower priority contexts. Eight of the contexts are assigned to the high-priority level, seven to the low-priority level, and one to background priority. The priority scheme and preemption allow the NPE to service time-critical functions such as providing data to prevent underflow on an HSS output port while carrying out longer latency tasks in the background. The NPE also has a set of 64 event inputs called start conditions. The start conditions are driven by the coprocessors and other functional units in the network processor, and any context can be swapped out under program control to wait upon any of the 64 start conditions. NPE memory includes 32 general-purpose registers, up to 8k instruction words, and up to 32k Bytes of data memory.

The NPE coprocessors are divided into two main categories, interface coprocessors and acceleration coprocessors. The interface coprocessors are used to implement both external interfaces to the NPU such as the HSS, Utopia, and Ethernet ports, and also to create internal connections to other parts of the NPU, such as interfacing to the internal busses of the NPU for memory access or for providing data to the media interface. This latter function is an important aspect of creating a "line card on a chip" programming model for the NPU. Packets and cells that are handled by the NPE for Layer 2 processing can be sent to the media interface of the NPU to be handled at Layer 3 and above by the microengines. The software running on the microengines does not need to be aware that the MAC or framer functions were handled on-chip, but can receive the data through the media interface in exactly the same manner as if the data were provided by an external Layer 2 IXF device.

NPE acceleration coprocessors, such as the encryption coprocessors, act as generally available resources for other compute elements in the NPU to utilize. Packets or cells to be encrypted are placed into a queue of data waiting to be processed, and the NPE fetches the data from memory through its internal chip interface coprocessors. It performs the encryption or decryption function with the encryption acceleration coprocessors and writes the results back to memory through the interface coprocessor again. This resource model is another key aspect of the line card on a chip, where the NPE acceleration coprocessors can process any packets enqueued to them from the microengines or from the Intel XScale® core. The packets do not have to have entered the system via the NPE interfaces.

Intel XScale® Core

The Intel XScale core is a 32-bit general-purpose processor that implements the industry-standard ARM* v5TE instruction set. Its role in the network processor includes initialization of the NPU, programming the microengines and NPEs, handling exception packets passed to it by the microengines or NPEs, and control plane processing. In a typical application it runs a general-purpose or realtime operating system such as Linux* or VxWorks*. Features that can be varied to meet the performance targets of different IXP NPUs include on-chip busses, writeback and writethrough caches, cache coherency with microengine and NPE writes, shared and dedicated memory interfaces, and clock speeds.

Combining the Elements in a 90nm IXP Processor

The IXP family of network processors takes advantage of these building blocks to achieve a wide range of packet processing performance. Intel's 90nm semiconductor process enables us to combine the building blocks to provide not only high-bandwidth packet processing, but also integrated features to create a true line card on a chip (Figure 2). A potential midrange architecture combines high-speed microengines for Layer 3+ packet processing, NPEs for Layer 2 and encryption processing, integrated Gigabit Ethernet MACs, and an Intel XScale core with a high-performance cache system. It would include memory and IO interfaces to appropriately support each of the included computing elements.

The line card on a chip approach provides cost-effective solutions for both wired and wireless communications. The combination of computing and connectivity resources will enable creation of low-cost, low-power cards. In the first example (Figure 3), a 90nm NPU is combined with low-cost framers, T1/E1/J1 LIUs, and Gigabit Ethernet PHYs to build a BSC/RNC line card implementing the Iub interface to the BTS/Node B. Reducing the number of T1/E1/J1 links and depopulating some of the off-chip memory makes the same architecture applicable to the BTS/Node B end of the link. A highly integrated NPU would allow the use of low-cost framers by performing the channelization of the T1/E1/J1 links and including an integrated HDLC controller.

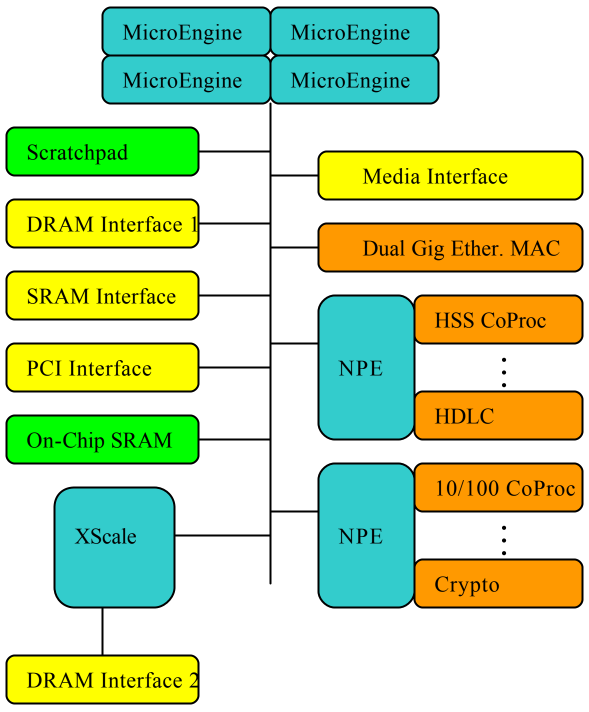


Figure 2: Possible 90nm NPU implementation

The second example design demonstrates how a highly integrated NPU could enable efficient solutions for wired communications (Figure 4). By gluelessly interfacing to up to eight DSL PHY chips, a single IXP processor could support up to 127 DSL ports. The connection from the DSLAM line card to the backplane would be via the integrated Gigabit Ethernet MACs. The Intel XScale core on the NPU can function as an integrated control plane processor, saving cost, power, and board space.

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

* Other names and brands may be claimed as the property of others

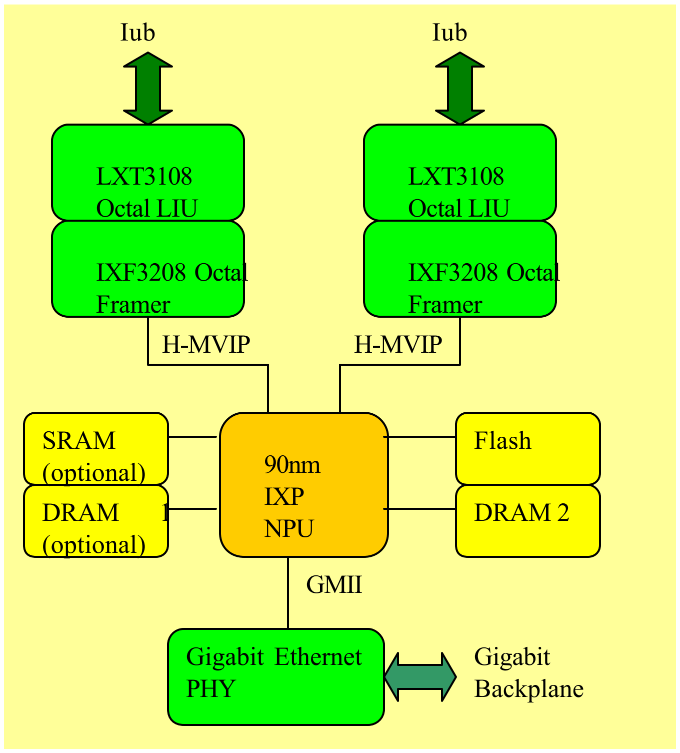


Figure 3: Radio Access Network line card

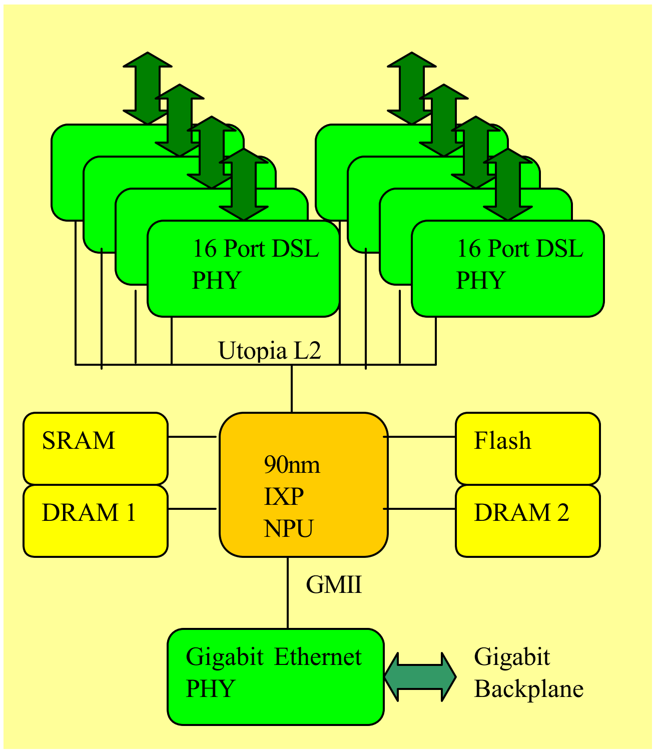


Figure 4: DSL line card

SCALABLE IXF ARCHITECTURE

The IXF family of devices provides the connectivity required to link Intel IXP processors to physical media. The IXF components in the Intel IXA cover a wide range of functions, protocols, and line rates [15].

- Ethernet-related IXF components include 10/100/1000 Ethernet MACs/PHYs, Single-/Multi-port devices and 10G MACs and Optical PHYs.
- SONET and SDH-related IXF components include optical modules, Phys, CDRs, SERDESs, and Framers in a wide range of line speeds ranging from OC-3 to OC-192.
- PDH-related IXF components include Single-/Multi-portT1/E1 LIUs and Framers.

Each IXF component supports industry-standard data and control interface. A component typically scales across a wide range of bandwidths and/or functions. A detailed review of all IXF silicon devices is beyond the scope of this paper. We focus on a specific IXF family, the next-generation 90nm IXF *link aggregator* family of devices.

Link Aggregator Family

A link aggregator consolidates processing for a wide range of Layer 1 protocols, SONET/SDH/PDH/VC termination [13], a wide range of Layer 2 protocols, and HDLC/POS/ATM/TDM [5], [8], [11]. The family of devices is all pin compatible and have the same footprint on a board. Each link aggregator offers several benefits of at the system level:

- Significant functional flexibility in the Layer 1 and Layer 2 protocols that can be used to process data
- Significant functional flexibility in the number and bandwidth range of logical Layer 1 and Layer 2 channels
- The provisioning and control of the functionality and performance of Layer 1 and Layer 2 features are orthogonal to one another
- Support for industry-standard line and network interfaces. Significant functional flexibility when combined with an IXP NPU on a line card.
- Support for priority-based service of latency-sensitive data such as voice and/or other data guaranteed a higher QoS.

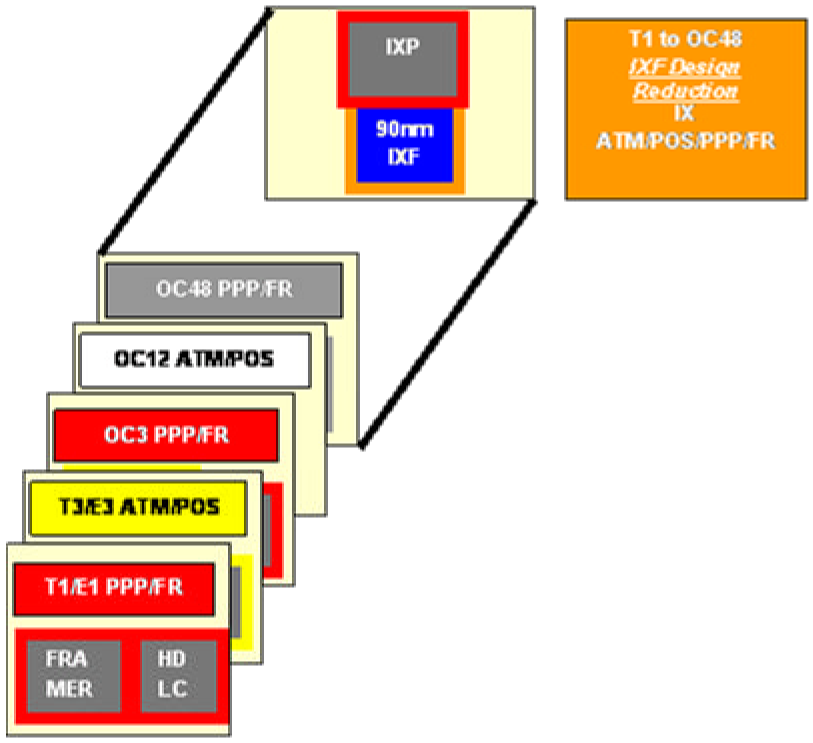


Figure 5: Scalability of a link aggregator-based line card

These are discussed in more detail below. The high-end link aggregator device operates at a cumulative bandwidth of OC-48 (2.488 Gbps) [13] and processes data from several thousand logically independent channels. The high-end link aggregator consolidates the functionality of several devices from other vendors in a single IC.

Link aggregators can be applied to a wide range of platforms at the edge of the network in the wireline and wireless infrastructure including multi-service switches, edge routers, 3G RNCs, and 2.5G BSCs. The devices are being developed with Intel 90nm fabrication technology [16]. Figure 5 shows a high-level view of the internal architecture of a link aggregator device. A number of architectural features enable the link aggregator family to scale across a wide range of functions and bandwidths, and these are described in the following sections.

Layer 1 Multi-Protocol/Multi-Linerate Support

The link aggregator family terminates SONET/SDH/PDH protocols. For SONET/SDH, each link aggregator supports a wide range of VT/VC/PDH options. Each link aggregator offers significant flexibility in the range of inputs that can be supplied to it under a maximum cumulative bandwidth constraint and supports built-in automatic protection switching on each input. The line inputs are industry-standard interconnects. The flexibility is realized by processing engines that scale in bandwidth and functionality through memory-based context-switched engines.

Layer 2 Multi-Protocol/Multi-Channel Rate Support

Every link aggregator can process data form each channel in a wide range of Layer 2 protocols. To enable flexible resource utilization and efficient implementation, Layer 2 protocols are processed on multi-protocol engines. Context-switched engines are again used to scale hardware across multiple protocols and bandwidths. A Layer 2 engine can process channels ranging in bandwidth from sub 64 Kbps upwards. Layer 2 engines can terminate Frame Relay [8], PPP [5], POS [13], or ATM [11] traffic in a channel. The packets output from the Layer 2 engines are output over one of two network interfaces supporting industry-standard interconnects. Channels may be assigned to the two network interfaces in a variety of configurations.

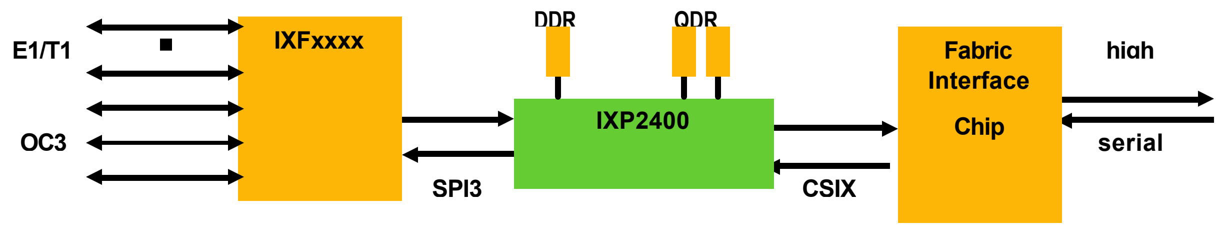


Figure 6: Single IXP2400-based edge line card

Multi-Link Protocol Support

Multi-link protocols, such as MLPPP/MLFR/IMA [6], [9], [10] that aggregate multiple channels to realize higher bandwidth virtual channels, are also fully supported by the Layer 2 engines. A link aggregator can simultaneously terminate traffic of different types in different channels. Link aggregators also support the transport of TDM traffic over packet/cell networks.

Flexible Layer 1/Layer 2 Channelization

Every link aggregator can fully channelize the total input bandwidth flexibly to channels ranging in speed up from 64 kbps (DS0/VT1.5) [13]. The channelization of the input traffic can be non-uniform. The entire bandwidth from the line inputs can process and output in a group of channels that may differ in individual bandwidths by more than a factor of 1000. The channelization is orthogonal to the actual physical inputs to the device.

A stream aligner is used to map data output from Layer 1 engines to Layer 2 engines. The mapping mechanisms in the stream aligner enable multiple Layer 2 engines to share hardware resources. The sharing enables Layer 2 engines to support the same diversity (in channel bandwidths) supported by the Layer 1 engines. The stream aligner also enables Layer 1 and Layer 2 provisioning and control to be orthogonal to one another.

Internal State Reduction

To efficiently realize a large number of multiple independent logical channels, the implementations were architected to scale through external memory. To relax performance and functional limits, especially at the high end, flow state for many functions was moved to external memory. The memory access bandwidth required by a channel will vary according to its data bandwidth. Priority-based schemes are used in Layer 1 and Layer 2 engines to guarantee required access to shared internal and external resources, such as external memory and the packet interfaces. When state is used internally, to further reduce internal memory sizes, protocol-specific techniques to compress internal state were developed.

Priority-Based Service

Each link aggregator offers hardware support for QoS for higher priority flows [7]. The hardware support minimizes jitter and end-to-end latency for these flows. This enables support for QoS-sensitive multi-media services in converged voice/data networks.

NPU-Based Function Enhancement

At the system (line card) level, additional value can be realized when the link aggregator is combined with an IXP network processor on the line card. For example,

- Egress traffic flow can be cooperatively shaped between the link aggregator and the IXP NPU. The link aggregator provides per-flow state information in-band over its packet interface to an external agent, such as an NPU.
- Protocols designed to enhance bandwidth efficiency for small packets such as Point-to-Point Protocol (PPP) Muxing [17] can be realized at the system level cooperatively across an IXP NPU and the link aggregator.
- The standards in TDM-over-IP (TDMoIP) protocols [14] are still evolving. TDMoIP functionality is implemented cooperatively across a link aggregator and an IXP NPU. By supporting TDMoIP services, the card will also support the convergence of voice and data networks in edge platforms.

Application Advantages

The link aggregator family offers several advantages at the system level:

- The link aggregator family significantly simplifies the design of line and service cards for edge platforms. At the high end, a simple design consisting of an IXF link aggregator and an IXP NPU replaces a complex design with a large number of components.
- The link aggregator family enables a significant reduction in the number of distinct logical designs (as shown in Figure 5). This simple design can process a wide range of line rates and functional protocols.

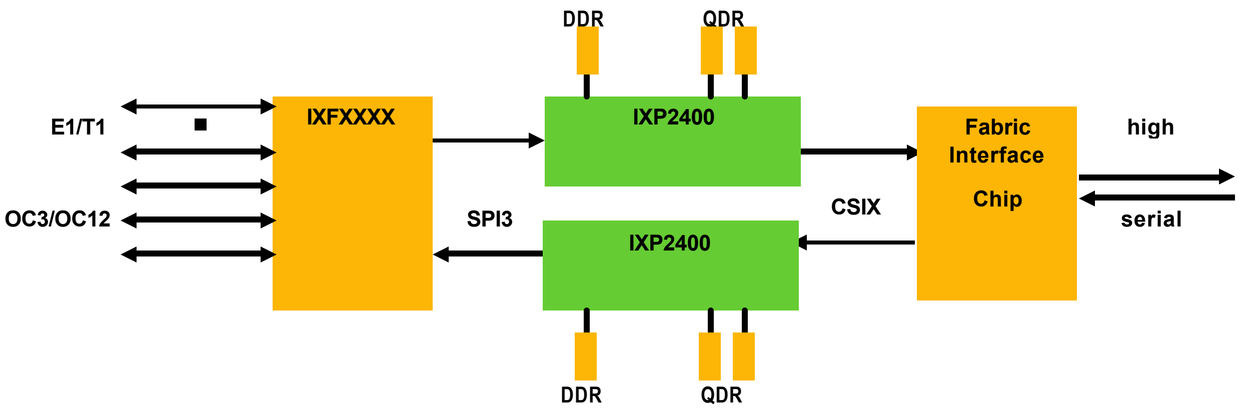


Figure 7: Dual IXP2400-based edge line card

- Significant integration enables the link aggregator family to provide superior physical parameters at each design point. In addition, integration enables in-field reliability (FIT rates) to be significantly improved. These advantages are captured in Table 2.

Table 2: IXF Link aggregators system-level benefits

Configuration	**Total Bitrate Mbps**	**No of Link Aggregators + Memory**	**Power <**
28 x DS1 inputs, 512 L2 channels	55	**1 + 4**	10W
1 x OC3 inputs, 2K L2 channels	155	1 + 4	10W
12 x DS3 inputs 4K L2 channels	155	1 + 4	15W
4 x OC12 inputs, 8K L2 channels	622	1 + 6	25W
1 x OC48 8K L2 channels	622	1 + 6	25W

- Unlike fixed function designs, the design can be adapted to changing traffic mixes. For example, when used in wireless networks, link aggregators can support an in-field migration from ATM to IP backhaul networks. As voice and data networks converge in edge platforms, link aggregators can enable the support of TDM traffic over IP networks.

MULTI-SERVICE SWITCH AND EDGE PLATFORMS

The input to an edge platform is data carried on SONET/SDH, PDH, or Ethernet lines. Line cards process the data on these inputs, terminate Layer 1 (SONET/SDH/Ethernet) and Layer 2 (ATM/FR/PPP) protocols and produce network-ready IP packets that are routed over the backplane to a trunk card in the same edge system.

Each line card has to process data from several thousand logically independent channels. The IXF framers provide the line side interface to aggregate E1/T1 traffic, DS3, OC-3, OC-12, and OC-48 traffic into a SPI-3 or SPI-4.2 pipe with aggregate data rates ranging from OC-12 to OC-48. These feeds support various protocols such as frame relay, TDM, ATM etc. The framer supports HDLC framing, multi-link PPP, multi-link FR, and ATM TC.

The following protocol processing blocks are representative of the application for these configurations:

- Frame relay to ATM interworking specified by FRF5, FRF8
- MPLS to ATM interworking as specified by IETF Martini draft
- ATM to IP interworking using AAL5

For each of these protocols, the processing blocks can be decomposed into functional blocks; each of these blocks performs certain tasks. Examples of functional blocks include Frame Relay receive, ATM AAL5 segmentation and reassembly, fabric segmentation and reassembly, protocol processing and inter-working, route lookup and classification, queue manager, congestion management, traffic policing, and scheduling.

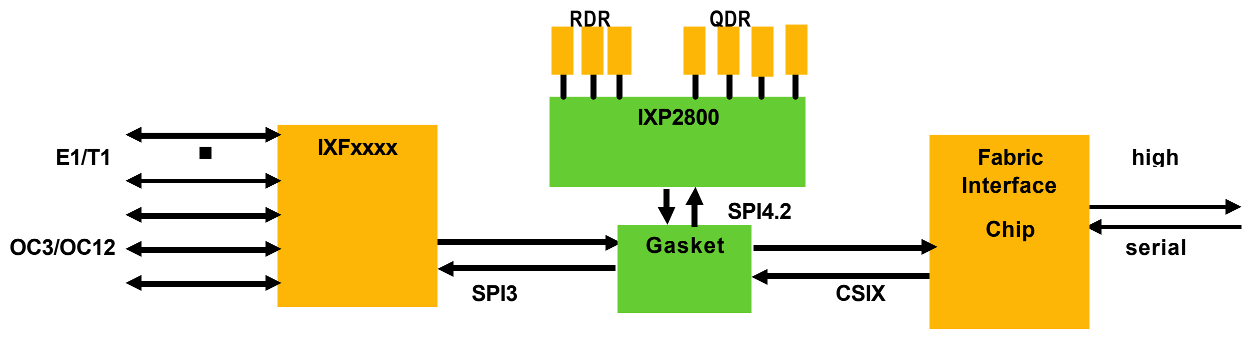


Figure 8: Single IXP2800-based edge line card

Different members of the IXP network processor family can be used in these line card configurations depending on the aggregate data rate required, the performance headroom required, and other line card design considerations such as Bill of Materials (BOM) cost, power, etc.

For this application, a single IXP2400-based line card shown in Figure 6 supports up to full-duplex OC-12 data rate. The eight microengines in IXP2400 running at 600 MHz provide sufficient horse power for this configuration

The same application is supported at OC-48 data rates by using two IXP2400 NPUs as shown in Figure 7. This solution provides a total horse power of sixteen microengines at 600 MHz. The application is partitioned such that the ingress processing is performed on one IXP2400 and the egress processing is performed on the second IXP2400.

Optionally, the same application can be supported on a single IXP2800-based line card. This solution provides significant performance headroom for future addition of services and processing blocks as the IXP2800 supports sixteen microengines running at 1 GHz. This approach eliminates the need to partition the application statically into ingress and egress blocks and provides flexibility on how the microengines are assigned for the various tasks. However, this line card requires an SPI-3 to SPI-4.2 gasket to interface with the IXF framer as shown in Figure 8.

As shown in the line card configurations above, the various members of the IXP network processor family such as the IXP2400 and the IXP2800 along with the IXF framer family provide a scalable and flexible solution to the multi-service switch and edge platforms.

CORE/METRO PLATFORMS

The core and metro infrastructure platforms are characterized by line card configurations such as 1-4 OC-48 ports, 1-10 Gigabit Ethernet ports, 1-2 OC-192 ports etc. This segment is expected to evolve in the future to support quad OC-192 configurations, 4x10G configuration as the demand for core/metro bandwidth increases. The Layer 2 transport supported in these platforms is typically POS, ATM, or Ethernet. The protocol processing typically involves TCP/IP, ATM, MPLS, ATM-IP, inter-working, etc.

The following protocol processing blocks are representative of applications running on the core/metro platforms:

- IPv4/IPv6 LPM-based route lookup
- IPv4/IPv6 5-tuple, 7-tuple packet classification
- Encap/Decap for the various L2 protocols
- Single-rate or dual-rate three-color marker metering and other policing algorithms
- Congestion avoidance algorithms such as Random Early Drop (RED) or Weighted Random Early Drop (WRED)
- Byte-count and packet-count statistics per flow, queue, or port
- ATM-IP inter-working, including ATM SAR
- MPLS label insertion/deletion, MPLS tunneling
- Sophisticated packet and cell domain scheduling and shaping functions

The specific applications and platform requirements vary widely with the performance target spanning the configurations described below:

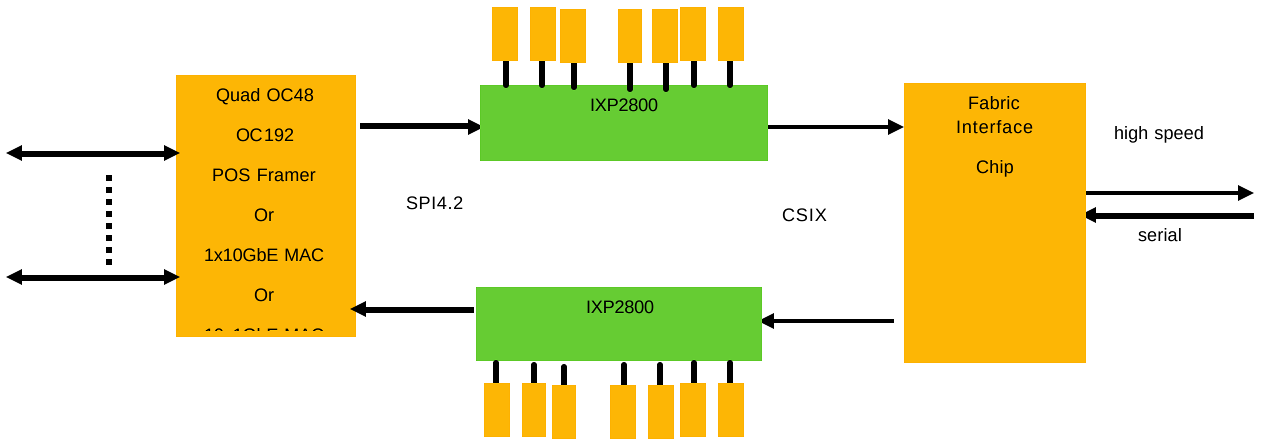


Figure 9: Dual IXP2800-based Core/Metro line card for low-touch applications

- Low-speed (1-2 OC-48 pipes, 4-5GbE ports), low-touch (simple L3 forwarding, simple protocol conversion, encap/decap, simple classification, minimal QoS) core/metro application
- Low-speed, high-touch (complex protocol inter-working, sophisticated L4+ based classification, congestion management, hierarchical QoS, fine-grained statistics) core/metro application
- High-speed (4xOC-48 pipes, 10GbE ports, 1xOC-192 pipe), low-touch core/metro application
- High-speed, high-touch core/metro application

The IXP2800 provides the basic scalable building block for the core/metro line cards. The IXP2800 offers two variants, with microengines running at 1 GHz and 1.4 GHz, respectively. These typically target the low-speed and high-speed segments, respectively. The faster NPU can also be used in the low-speed, high-touch segment to handle the deep packet inspection requirements. Multiple IXP2800 chained together in ingress and/or egress direction offer the necessary horse-power for a high-speed, high-touch application.

A family of SPI-4.2-compatible optical framers and Gigabit MAC chips from the IXF family enable a class of scalable line cards to meet these wide-ranging speed requirements.

Figure 9 shows a typical line card configuration for a low-touch application. The media interface chip is either a POS framer or a GbE MAC that supports the SPI-4.2 interface. A 1 GHz or a 1.4 GHz IXP2800 can be used in the line card, depending on the speed and on the number of ports supported on the interface.

Figure 10 shows the line card configuration for a high-touch application. Multiple IXP2800s can be used in either ingress and/or egress directions to provide the necessary compute and bandwidth horsepower required for the deep-packet inspection, high-touch application.

MAPPING PROCESSING BLOCKS TO MICROENGINES

The functional code for the various blocks listed in the above sections is identical for the various members of the IXP network processor family as they use the same instruction set. Based on the speed of the incoming port and the total number of ports supported, these blocks are mapped to the various microengines at different frequencies, and they use different resources to achieve the required target.

The compute budget available per microengine (typically represented in terms of the number of microengine cycles) for processing any incoming packet depends on the following parameters:

- Minimum packet size and the worst-case packet arrival rate for a given line rate, e.g., OC-192 min packets arrive 40ns apart
- The frequency of the microengine

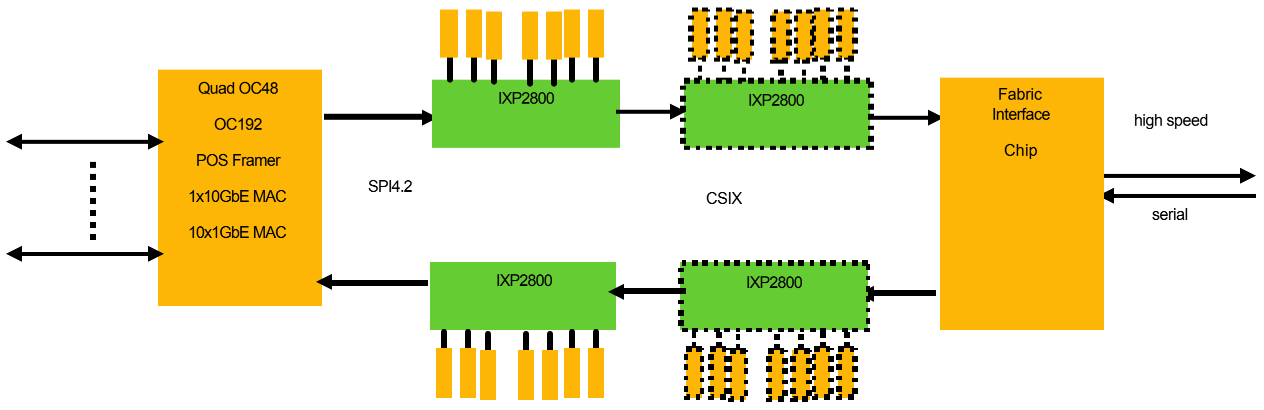


Figure 10: Multiple IXP2800-based Core/Metro line card for high-touch applications

The total number of cycles required for an entire application is determined by the number of functional blocks in the application and the cycles required per functional block. Parallelization and mapping of functional blocks to available microengine resources depends on the characteristics of the blocks.

- If the functional block does not have a critical section such as a read-modify-write (RMW) variable that requires exclusive access to only one agent at a time, microengines can be assigned in pools to perform the task. In other words, if a block requires M cycles and each microengine has a budget of N cycles, M/N microengine are assigned to the pool to meet the requirement. Examples of such blocks include LPM-based route lookup, 5-tuple or 7-tuple classification, etc.

- If the functional block includes a critical section, the locking mechanism for mutual exclusion can be implemented either local to the microengine or in global shared memory. The performance of global locking is dictated by the RMW latency to shared memory, and it does not scale to meet requirements for mutual exclusion at OC-48 and above. The features provided in the microengine such as CAM, local memory, and generalized Event Signaling are used to implement mutual exclusion mechanisms that scale up to OC-192 data rates. Examples of functional blocks that require mutual exclusion include all functions that require arrival order of packets/cells to be maintained such as queue management, CRC residue computation, metering, etc.

The next step after the application is mapped to the available microengine resources is to tune the performance to meet the requirements. The performance tuning involves the following three steps and is aided by a rich set of tools (compilers, profilers, simulation, and visualization tools).

- Optimizing the compute cycles for a given task to fit within the available number of microengine cycles. This involves reducing microengine idle cycles by eliminating NOP, reducing unfilled branch defer slots, reducing microengine stalls due to FIFO full condition, etc.

- Optimizing the code for IO latencies such that the microengine does not idle waiting for data to return from external memory. This involves using the hardware threads, pipelining the hardware threads further by phasing techniques to increase the number of packets handled concurrently, distributing and load balancing the traffic across the available internal buses to optimize utilization and minimize queuing delays.

Optimizing the performance of inter-thread token passing within a microengine and/or across microengines such that critical sections mapped to a single engine or multiple microengines fit within the available budget.

CONCLUSION

The Intel IXP network processors and IXF framers and link aggregators combine to make Intel IXA highly flexible and scalable. With a single architecture these components can create line cards that perform cost effectively from T1/E1/J1 rates for 3G cellular base stations and metro edge aggregation up to OC-192 rates for core routers. With the modular computing architecture of the IXP network processors, the same software can be used across all these data rates, providing development cost savings and fast time-to-market for network equipment providers.

REFERENCES

[1] Williams J., "Net Growth Pushes Router Limits," *EE Times*, Issue 1083, October 1999.

[2] Mangione-Smith, W. and Memik, G., "Network Processor Technologies," *Tutorial*, MICRO 34, December 2001.

[3] Ginsburg, D. and Hattar, M., *Implementing IP Services at the Network Edge*, Pearson Education, New Jersey, 1999.

[4] Carlson, B. and Adiletta, M., *Intel Internet Exchange Architecture and Applications: A Practical Guide to Intel's Network Processors*, Intel Press, June 2003.

[5] Simpson, W., "The Point-to-Point Protocol (PPP)," *STD 50*, RFC 1661, *Internet Engineering Task Force,* July 1994.

[6] Sklower, K., Lloyd, B., McGregor, G., Carr, D., and Coradetti, T., "The PPP Multilink Protocol (MLPPP)," RFC 1990, *Internet Engineering Task Force,* August 1996.

[7] Bormann, C., "The Multi-Class Extension to Multi-Link PPP," RFC 2686, *Internet Engineering Task Force*, September 1999.

[8] Sinricope, D., Editor, "User-to-Network Implementation Agreement," FRF 1.1, *The Frame Relay Forum*, January 1996.

[9] Mattson G., Editor, "End-to-End Multilink Frame Relay Implementation Agreement," FRF 15, *The Frame Relay Forum*, August 1999.

[10] Vallee, R., Editor, "Inverse Multiplexing for ATM (IMA) Specification Version 1.1," AF-PHY-0086.001, *ATM Forum*, March 1999.

[11] ITU, "B-ISDN ATM Layer Specification," ITU-T *Recommendation I.361*, February 1999.

[12] Forouzan, B.A., *TCP/IP Protocol Suite*, McGraw-Hill, Boston, MA, 1999.

[13] Kartalopolous, S.A., *Understanding SONET/SDH and ATM: Communications Networks for the Next Millennium*, Wiley-IEEE Computer Society, 1999.

[14] Stein, Y., Shaashoua, R., Insler, R., and Anavi, M., "TDM over IP," draft-anavi-tdmoip-05, *Internet Engineering Task Force*, March 2003.

[15] Ambrose, A. and Llorens, H., Editors, *Intel Communications Solutions Manual Version 4*, www.intel.com

[16] Intel Press Release, "Intel Unveils World's Most Advanced Chip-Making Process," www.intel.com/pressroom/archive/releases/20020813tech.htm

[17] Pazhyannur, R., Ali I. and Fox C., "PPP Multiplexing," RFC 3153, *Internet Engineering Task Force,* August 2001.

AUTHORS' BIOGRAPHIES

Bapi Vinnakota is a principal engineer in the Intel Communications Group. He received a B. Tech degree from the IIT, Madras, India and a Ph.D. degree from Princeton University in 1991, both in Electrical Engineering. He was on the faculty of the Department of Electrical Engineering at the University of Minnesota from 1991-2001. He has been with Intel Corporation since 2001. His technical interests include communication and networking ICs and CAD. His e-mail is bapi.vinnakota at intel.com.

Paul Dormitzer is an architect in the Network Processor Division. He previously worked at Digital Equipment Corporation for ten years as an architect and ASIC designer for VAX and Alpha systems, and for the past five years he worked at ADC Telecommunications designing cable modem termination systems. He holds a B.A. degree in Computer Science from Harvard University. His e-mail is paul.h.dormitzer at intel.com.

Mark Rosenbluth is an architect in the Network Processor Division. He has been at Intel for five years and prior to that worked at Digital Equipment Corporation, where he was architect for PCI Bridges and also worked on VAX and Alpha microprocessors. He received a B.S.E.E. degree from Rutgers University. His e-mail is mark.rosenbluth at intel.com.

Sridhar Lakshmanamurthy is a senior staff architect at Intel's Network Processor Division, focusing on the definition of next-generation network processors and performance analysis of Intel's network processor solutions. Prior to joining the Network Processor Division, Sridhar focused on platform performance analysis for Intel server chipsets for Xeon and Itanium Product Family (IPF) processors, and system bus specification for the IPF processors. Sridhar joined Intel in 1993 after receiving an M.S. degree in Computer Engineering from Rice University in Houston, TX. He also holds a B.E. degree in Electronics and Communication Engineering from Osmania University, Hyderabad, India. His e-mail is sridhar.lakshmanamurthy at intel.com.

For further information visit:
developer.intel.com/technology/itj/index.htm