Implementation of network systems using network processor technology: performance evaluation

Koert Vlaeminck*, Tim Stevens†, Wim Van de Meerssche, Filip De Turck, Bart Dhoedt, Piet Demeester
Department of Information Technology, Ghent University - IMEC
Sint-Pietersnieuwstraat 41, B-9000 Gent, Belgium.
Tel.: +32 9 331 49 42, Fax: +32 9 331 48 99
E-mail: koert.vlaeminck@intec.ugent.be*, tim.stevens@intec.ugent.be†

Abstract— **This paper presents the design of a network system ng network processor technology. Firewall / network address id port) translation was selected as reference application and el's second generation network processor (more specifically : IXP2400) as target platform. Implementation details of the vice reveal some typical characteristics of network processor ogramming. A detailed performance analysis of the imple- nted services is compared to coarse analytical models and system's bottleneck is identified, while possible solutions to viate this bottleneck are suggested.**

I. Introduction

The current tendency to increase application awareness l intelligence in IP network nodes, in order to meet sub- iber demands, especially in residential access environments, arly necessitates the presence of additional packet process- units in network nodes. Due to their specific hardware hitecture—based on far-reaching parallelism features—and ir low power consumption, network processors are a perfect ch for upgrading current networking equipment.

his paper presents application architecture and perfor- ice details for a firewalling / network address (and port) slation (NAPT) service implemented on one of Intel's ond generation network processors, the IXP2400. A fire- ling / NAPT service is selected because it offers a clear simplified use case of a more generic service enabler: ket classification. First, a pipelined architecture similar to ix' Netfilter is mapped onto the available hardware re- ces, revealing typical characteristics of network processor gramming. Later, the implemented solution is evaluated compared to coarse analytical models, identifying the em's bottleneck is processing power rather than memory ncy, which is reduced by the parallelism features. Possible nizations are highlighted.

ne remainder of this paper is structured as follows: Sec- II reports on related work in this area. Section III presents ort introduction to access networks, where the network ems discussed in this paper can be deployed on a large e. The selected reference services, firewalling and NAPT, her with their mapping to the selected hardware, are red in section V, while Section VI presents a detailed per- ance analysis, including coarse analytical models. Finally, on VII presents possible future work and concluding rks.

II. Related work

Design and development of network systems using network processors (NPU) is a hot topic. In August 2003, a special issue of IEEE Network was dedicated to the subject [1]. Two articles were published providing an in-depth description of how specific data plane functions can be implemented using an NPU: one article on the implementation of a DiffServ Edge router on an a first generation Intel NPU [2], the IXP1200, the other on the implementation of an IPv4/IPv6 transition mechanism on that same NPU [3]. Another article in that issue describes the design and implementation of an intelligent DSLAM using NPUs [4]. For the data plane functionality, again the IXP1200 was selected.

More recently, the October 2004 issue of IEEE Micro was dedicated to network processors. In this issue, we still see papers on the first generation Intel network processor: e.g. [5] presents a simulation infrastructure for the IXP1200 which also provides an estimation model for measuring the simulated processor's power consumption. However, we also see the second generation Intel network processors appear: e.g. [6] presents the PRO3 hybrid NPU architecture and compares it to the Intel IXP2400, while [7] compares, analyzes and optimizes cryptographic algorithms on the Intel IXP2800.

Similar work on the implementation of service enablers on content processors, such as the Broadcom SB1250 (as opposed to network processor, like the aforementioned Intel IXP families), was presented in [8].

III. An evolution towards IP-aware access Networks

Current (DSL) access networks are often ATM (Asynchro- nous Transfer Mode)-based, where end-user devices set up PPP (Point to Point Protocol) connections inside ATM VCs (Virtual Circuits) to the Broadband Access Server (BAS) at the edge of the network. Only the Customer Premises Equipment (CPE) and the BAS are IP-aware, while the (ATM) aggregation network operates at Layer-2 of the OSI stack. The Access Multiplexer's (DSLAM) sole task is the aggregation of users. Many carriers are exploring ways of migrating their ATM ac- cess infrastructure to an Ethernet-based access infrastructure. Most DSL deployments today transport Ethernet frames on top of ATM between the customer premises and the BAS any- way. Access networks based on Ethernet are claimed cheaper

and easier to install and maintain [9]. Migration towards an Ethernet-based access infrastructure and problems related to the inevitable ATM / Ethernet coexistence are out of the scope of this paper. We assume a (future) full Ethernet access

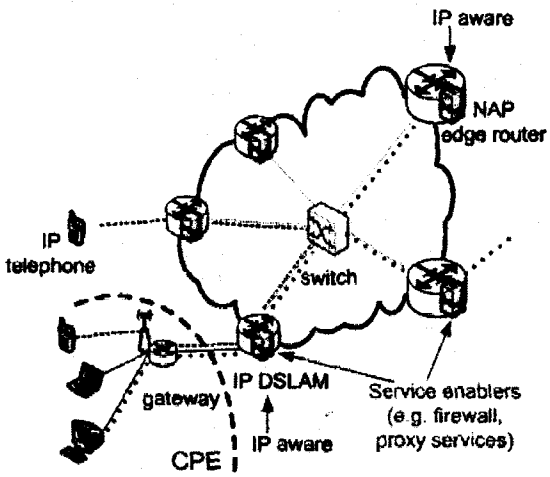


Fig. 1. Assumed access network topology: a full Ethernet access network with IP-aware access multiplexers (CPE: Customer Premises Equipment, DSLAM: DSL Access Multiplexer, NAP: Network Access Provider).

network with IP-aware access multiplexers. This implies IP-awareness at the CPE, the DSLAM and the Edge Router (ER), as depicted in Fig. 1. Future DSLAMs or ERs may offer a number of service enablers, or even implement some value-added services (e.g., firewalling & network address translation, intrusion detection, proxy services). Because of their flexibility, excellent packet processing capabilities and low power consumption, network processors (NPU) seem the right hardware for this task.

IV. Intel's Second Generation Network Processor

We selected Intel's second generation network processor (IXP2xxx) as a target platform for implementing our network system. Our evaluation board from Radisys, the ENP-2611 [10] depicted in Fig. 2, is a PCI board sporting an Intel IXP2400 network processor, 16 MB of flash memory to store boot code, 8 MB of Quad Data Rate (QDR II) SRAM, 256 MB of Double Data Rate (DDR) DRAM and a SPI-3 (System Packet Interface) Bridge FPGA, connecting three Gigabit Ethernet (GbE) interfaces to the IXP's Media and Switch Fabric (MSF) interface.

The major IXP2400 processing units are the XScale control processor and eight microengines (ME), organized into two clusters of four. The Intel XScale core is a general-purpose 32-bit RISC processor, compatible to the ARM Version 5STE Architecture [11], running at 600 MHz. It has a 32 KB data cache, a 32 KB instruction cache and a 2 KB mini-data cache. Its main functions are chip management and initialization, but it can also be used for higher layer network processing tasks.

The microengines are small RISC processors, also running at 600 MHz, with an instruction set specifically tuned for processing network data and support for eight hardware threads. Each ME has an independent instruction store large enough for 4K, 40-bit instructions, which is initialized by the XScale core. Furthermore, each ME has 640 long-words (= 8 bytes) of low-latency local memory. Microengines do the main data plane processing per packet.

The IXP2400 also has a Hash Unit, offloading hash calculations from the XScale core and the microengines. Furthermore it has support for three types of memory, shared between the different processing units. The 16 KB of on-chip scratchpad memory is the fastest. Two separate SRAM controllers provide high-speed access to up to 128 MB of QDR SRAM (64 MB per channel). SRAM is typically used for control information storage. Finally, the DRAM controller provides access to up to 1 GB of DDR DRAM, which is typically used for data buffer storage.

V. Programming an IXP2xxx

Intel second generation network processors are programmed using the receive - process - transmit paradigm [12] as depicted in Fig. 3: receive, process and transmit tasks run on different processing units (microengines and / or XScale core) with queues between them. The microengines can run a series of sequential processing tasks (pipelined approach) or a pool of parallel processing tasks or a mixture of both.

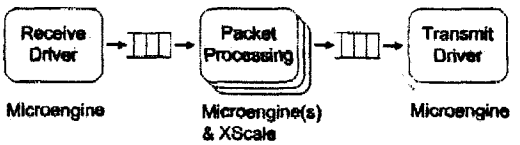


Fig. 3. Programming the IXP2xxx: the receive - process - transmit paradigm.

We selected firewalling and network address & port translation (NAPT) services as reference applications for implementation on the IXP2400. Both services are deployed on a large scale in the access network, though nowadays almost exclusively in customer premises equipment (CPE), requiring advanced end user networking skills. In future scenarios, it is likely that access nodes will implement such services to meet or anticipate customer requests. For network access providers (NAP), this will generate new revenue streams and possibly increased control over the data traveling through their networks.

Furthermore, firewalling / NAPT service offers a clear and simplified use case of a more generic service enabler: packet classification. Performance evaluation of a firewall / NAPT implementation on Intel network processor hardware gives a good indication of how other network services will perform on this architecture, since it is exactly this packet classification that induces a bottleneck on performance, as we will show in the next section.

We based our implementation on the netfilter / iptables [13] forwarding chain. Netfilter and iptables are building blocks of a framework that enables packet filtering, network address (and port) translation and other packet mangling inside the Linux kernel. Input and output chains process packets destined to

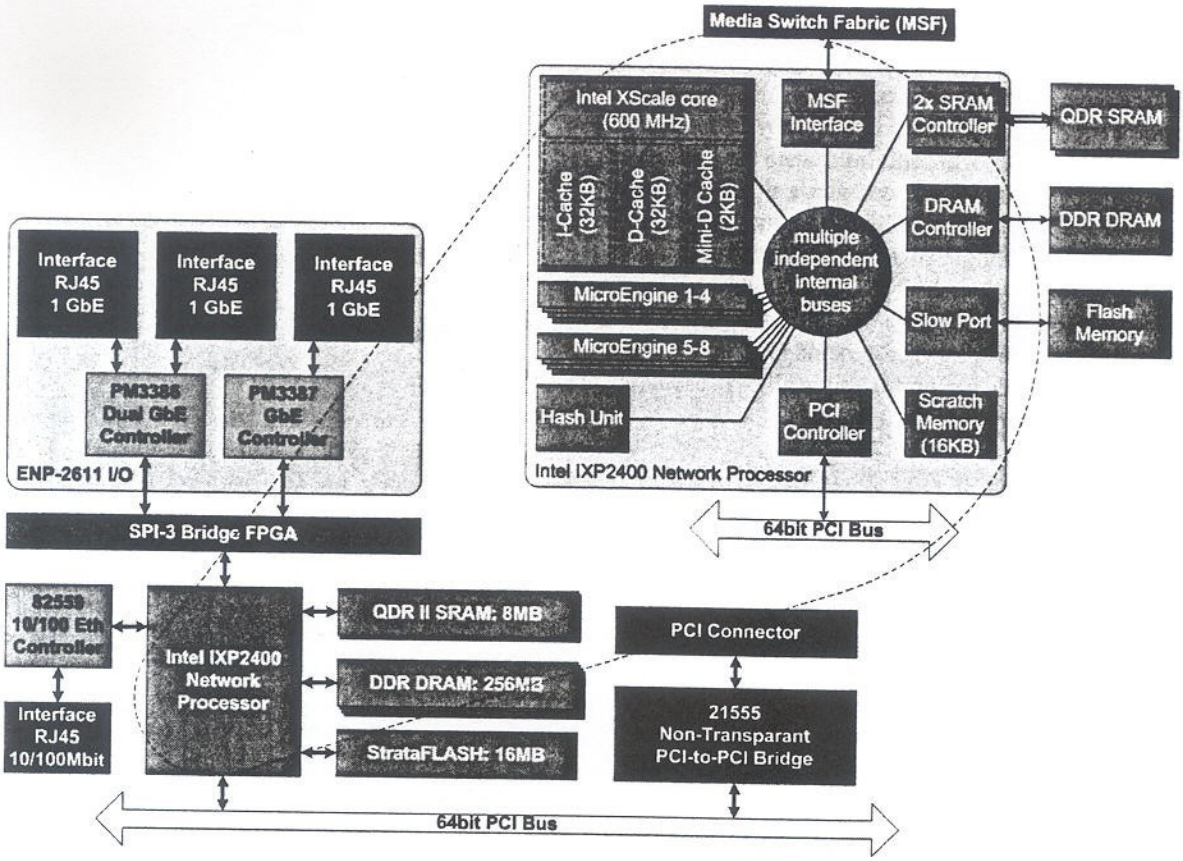


Fig. 2. Radisys ENP-2611 Intel IXP2400 evaluation board.

and originating from a host. As our firewall / NAPT service is meant for integration in access network equipment, only the forwarding chain is important for data plane traffic.

The netfilter forwarding chain consists of several hooks, which identify the sequential packet processing tasks:

(i) First, the pre-routing hook performs connection tracking and destination address / port translation;
(ii) Then the routing decision is made;
(iii) In the forwarding hook it is decided whether the packet is to be forwarded or not (firewalling decision);
(iv) Finally, the post-routing hook performs source address / port translation.

Connection tracking is fundamental for network address (and port) translation: packets belonging to the same connection have to be treated the same way. It also enables stateful packet inspection. A stateful firewall not only increases security, but packet processing speed may also increase, since the sequential search through the firewall rules, can be avoided for packets belonging to a registered connection.

Mapping the netfilter components to the IXP2400's processing units is relatively straightforward, as depicted in figure 4. Receive and transmit code each occupy one microengine. A third microengine is used for resource management:

- Thread 0: Allocate memory
- Thread 1: Free memory
- Thread 2: Search for conntracks that timed out

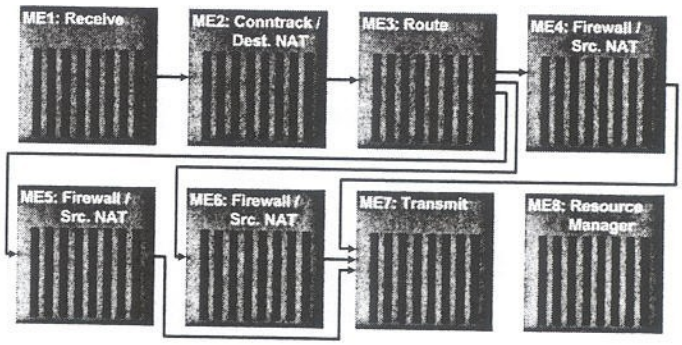


Fig. 4. Mapping of the firewall components on IXP2400 hardware.

- Thread 3: Allocate TCP ports for NAPT
- Thread 4: Allocate UDP ports for NAPT
- Thread 5: Allocate ICMP ID's for NAPT

This leaves five microengines for packet processing: the pre-routing code occupies one, routing occupies a second and firewall & post-routing can be combined on a third. Earlier experience [14] with Intel's first generation network processor (the IXP1200) has learned that the sequential search through the firewall rules is very resource intensive and induces a performance bottleneck. A first implementation, where packets were matched against the firewall rules at the core, only allowed 1500 new connections to be set up per second on

IXP1200 hardware[1]. Implementing the search through the firewall rules at the microengines increased this number to up to 5000 new connections per second for a small rule set. However, the firewall performance drops very fast with an increasing number of rules. This still holds true for the second generation of Intel network processors, as we will show in the next section. Therefore the firewall & post routing code was duplicated on the remaining two microengines. The routing microengine pushes the packets it has processed on a ring, which the three firewalling & post routing MEs use to get their packets. Packets can be processed independent of each other. That way, three MEs perform the firewall & post routing in parallel, each on different packets.

VI. Network Processor Performance Evaluation

A Spirent Smartbits 6000 network performance analysis system [15] was used to measure the IXP's performance. Each test was done using minimum sized IP packets (64 byte), implying a packet rate of 1,448,095 packets per second at GbE speeds. The packet inter-arrival-time is 672 ns, which means each microengine, running at 600 MHz, has 403 clock cycles per packet to do its work.

As already stated in the previous section, the sequential search through the firewall rules limits the IXP's performance. In stateful firewalls however, packets of known connections can be immediately accepted or rejected, based on connection tracking information, and only the first packet of each connection has to be matched against the complete firewall rule set. That way we were not able to stress our IXP2400 evaluation board using two Gigabit Ethernet interfaces (one connected to the network to protect, the other to an untrusted network). Therefore, for performance evaluation purposes, we forced the firewall to match every packet against its complete rule set. We measured the IXP's throughput for an increasing number of firewall rules. 100% means all 1.448 million packets / s can be processed. Results are summarized in Fig. 5 and will be explained below.

With only one firewalling microengine, our IXP2400 evaluation board is able to handle up to five firewall rules at gigabit speed. This is clearly insufficient. Duplicating the firewall code on two extra microengines (three firewalling MEs instead of one) just about triples the performance: up to 14 firewall rules can be handled at gigabit speed. This proves the sequential search to the firewall rules is indeed a bottleneck to the IXP's performance.

Although a multithreaded design – remember that each IXP2400 ME has support for eight hardware threads – helps in hiding the latency of the SRAM accesses required for fetching the firewall rules from memory, faster memory could save some cycles per firewall rule when processing a packet. We made an implementation where the rule set is stored in the firewall microengines' local memory. Data can be read from this memory in only three cycles (cf. 90 cycles for SRAM access). This implementation was able to handle 23 rules per packet at gigabit speed, using three MEs. On the downside, we were only able to store up to 80 rules in the limited – 640 longwords – amount of local memory.

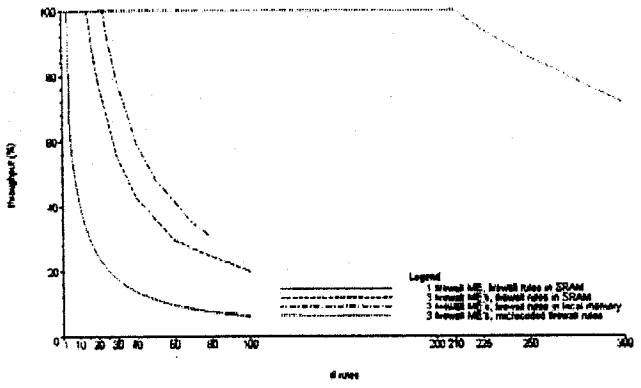


Fig. 5. Performance of the firewall / NAPT service on an Intel IXP2400 board, measured using two GbE interfaces (one for upstream, one for downstream) and minimum sized packets. Several versions of the flexible implementation presented in section V (with firewall rules in memory) are compared to an implementation where firewall rules are hand-coded and optimized in microcode (no memory accesses).

The ability to handle up to 23 firewall rules per packet at gigabit speed might not seem spectacular, but remember that in real-life operation, only the first packet of each connection has to be matched against the firewall rules, while for performance evaluation purposes we forced the firewall microengine(s) to process all packets. Furthermore there is still some room for improvement. To see how far we could push the IXP2400 hardware performance-wise, we hand-coded and optimized firewall rules in microcode. This implementation exploits no memory accesses and is very performing: up to 210 rules per packet could be handled at gigabit speed. Of course it is not possible to deploy this implementation for real-life operation, since updating the firewall rule set implies reimplementing and reoptimizing the microcode. However, it shows that using other algorithms for implementing the firewall – more optimal than a sequential search through the rule set and more flexible than hand-coding the firewall rules in microcode – could further increase performance, since there's clearly some headroom before the other components in the implementation could become a bottleneck.

Based on the fact that each microengine of the IXP2400 has 403 cycles per packet to do its work, we can estimate the performance of our firewall application. Both implementations – firewall rules in SRAM and firewall rules in local memory – require about 100 cycles per packet, independent of the number of rules on the firewall / post-routing ME. Furthermore the 'SRAM' implementation requires about 50 cycles per firewall rule per packet, while the 'local memory' implementation requires about 40 cycles per firewall rule per packet. This implies the 'SRAM' implementation can handle about 6 rules per packet per firewall ME, while the 'local memory' implementation can handle 8 rules per packet per

[1]Note that due to connection tracking, only the first packet of each connection has to be matched against the firewall rules.

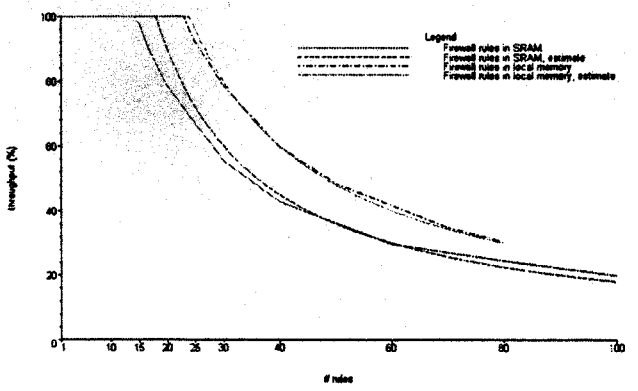


Fig. 6. Comparison of the actual IXP performance when running the firewall / NAPT service to a theoretical estimate, assuming three firewall MEs, each having a budget of 403 clock cycles per packet.

firewall ME. When we try to process more rules per packet, throughput performance will drop inversely proportional with the number of rules. These estimations, assuming three firewall MEs, are depicted in Fig. 6, and compared to the actual measured performance.

While the measured performance of the 'local memory' implementation is very close to this estimation, the 'SRAM' implementation clearly deviates from the estimated performance. This is due to the fact that this implementation requires multithreading to hide the latency of the SRAM accesses. Although the IXP2400 has support for 8 hardware threads, this still imposes some overhead, which was not taken into account in our very simple model. However, even the performance of the 'SRAM' implementation still shows the same tendency as the estimation.

VII. Conclusion and Future Work

In this paper we presented the design of a network system using network processor technology. We chose firewall / network address (and port) translation as our reference application and Intel's second generation network processor as target platform. Discussion of the implementation of this service on an Intel IXP2400 revealed typical characteristics of network processor programming. A detailed performance analysis of the developed application showed the sequential search through the firewall rules to be a bottleneck. Our very flexible – since rules can be updated in memory while the firewall is running – netfilter based implementation was able to handle up to 23 rules per packet at gigabit speeds using three microengines, checking every single packet. While totally inflexible, since updating the rule set implies reimplementing the firewall, the implementation where rules were hand-coded and optimized for execution on the microengines shows there is still a lot of room from improvement.

We are currently investigating other algorithms for implementing the firewall, more performing than an a sequential search through a rule set stored in memory and more flexible than hand-coding the rules in microcode. First results of an implementation based on an algorithm using ordered binary decision diagrams for representing the rule set, which allows larger rule sets to be used without sacrificing performance [16] look very promising and will be reported upon in a future publication.

References

[1] H. Vin, R. Yavatkar, *Guest Editorial: Network Processors*, IEEE Network, July/August 2003, Vol. 17, No. 4, Pages 10-11.

[2] Ying-Dar Lin, Yi-Neng Lin, Shun-Chin Yang, Yu-Sheng Lin, *DiffServ Edge Routers over Network Proccessors: Implementation and Evaluation*, IEEE Network, July/August 2003, Vol. 17, No. 4, Pages 28-34.

[3] E. Grosse, Y. N. Lakshman, *'Network Processors Applied to IPv4/IPv6 Transition*, IEEE Network, July/August 2003, Vol. 17, No. 4, Pages 35-39.

[4] R. Neogi, K. Lee, K. Panesar, J. Zhou, *Design and Performance of a Network-Processor-Based Intelligent DSLAM*, IEEE Network, July/August 2003, Vol. 17, No. 4, Pages 56-62.

[5] L. Yan, J. Yang, L. N. Bhuyan, L. Zhao, *NePSim: A Network Processor Simulator with a Power Evaluation Framework*, IEEE Micro, September/October 2004, Vol. 4, No. 5, Pages 34-44.

[6] I. Papaefstathiou, e. a. , *PRO3: A Hybrid NPU Architecture*, IEEE Micro, September/October 2004, Vol. 4, No. 5, Pages 20-33.

[7] Z. Tan, C. Lin, H. Yin, *Optimization and Benchmark of Cryptographic Algorithms on Network Processors*, IEEE Micro, September/October 2004, Vol. 4, No. 5, Pages 55-69.

[8] T. Vermeiren, E. Borghs, B. Haaodorens, *Evaluation of software techniques for parallel packet processing on multi-core processors*, IEEE CCNC 2004, Las Vegas, January 2004.

[9] L. Zier, W. Fischer, F. Brockners, *Ethernet-Based Public Communication Services: Challenge and Opportunity*, IEEE Communications Magazine, March 2004, Vol. 42, No. 3, Pages 88-95.

[10] Radisys Corporation, *ENP-2611 Hardware Reference*, August 2003, `http://www.radisys.com/files/support_downloads/007-01419-0003.ENP-2611HW.pdf`.

[11] *The ARM Instruction Set Architecture*, `http://www.arm.com/products/CPUs/architecture.html`.

[12] E. J. Johnson, A. R. Kunze, *IXP2400/2800 Programming, The Complete Microengine Coding Guide*, Intel Press, 2003.

[13] *Netfilter documentation*, `http://www.netfilter.org/documentation`.

[14] K. Vlaeminck, T. Stevens, F. De Turck, B. Dhoedt, P. Demeester, *Deployment of network processors in access networks to provide service enabling functions: evaluation results*, 9th European Conference on Networks & Optical Communications (NOC2004), June 29-July 1, 2004, Proceedings, Pages 70-77.

[15] *Spirent Smartbits 6000 Product Overview*, `http://www.spirentcom.com/documents/39.pdf`.

[16] S. Hazelhurst, A. Attar, R. Sinnappan, *Algorithms for Improving the Dependability of Firewall and Filter Rule Lists*, International Conference on Dependable Systems and Networks (DSN 2000), June 25-28, 2000.

ISBN [illegible]
Printed in the United States of America

PROCEEDINGS OF THE 2005 INTERNATIONAL CONFERENCE ON COMPUTER DESIGN

CDES'05

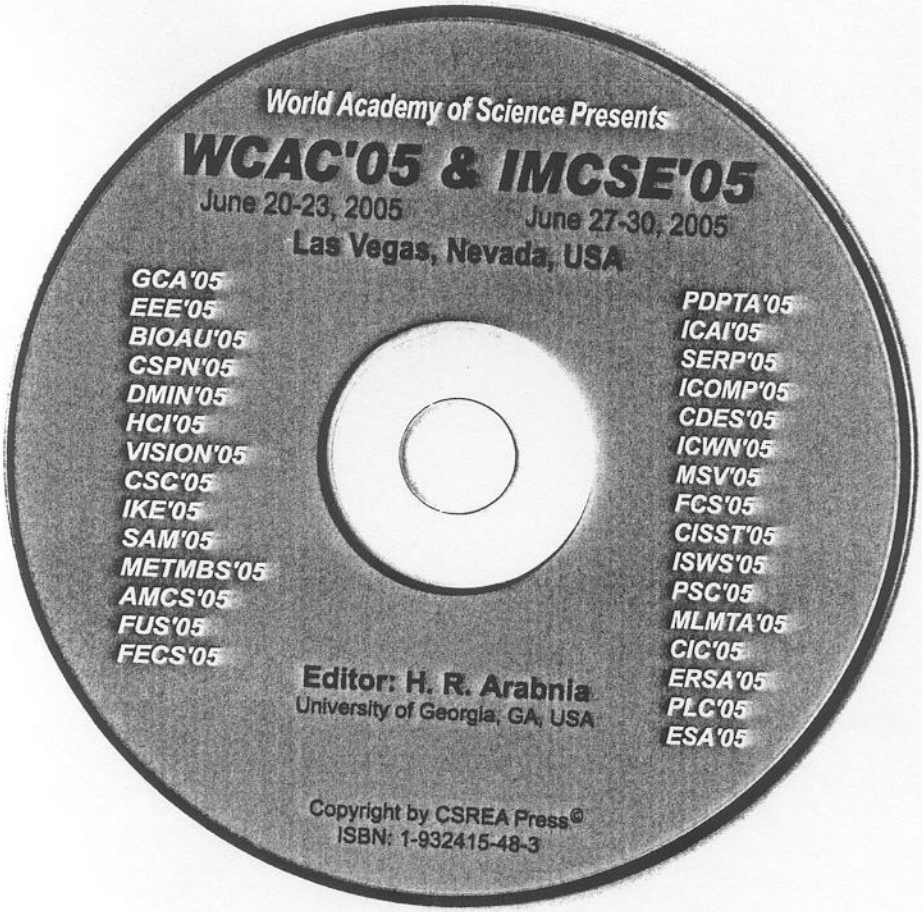
Editors

Laurence T. Yang, Hamid R. Arabnia
Yiming Li, Salam N. Salloum
Jose G. Delgado-Frias

Associate Editors

Dimitrios Soudris, Ben A. Abderazek, Antonio Pescape
V. Lakshmi Narasimhan, Mahir S. Ali, Hsun-Jung Cho
Aleksey Urmanov

Las Vegas, Nevada, USA
June 27-30, 2005


World Academy of Science Presents
WCAC'05 & IMCSE'05
June 20-23, 2005
June 27-30, 2005
Las Vegas, Nevada, USA
GCA'05
EEE'05
BIOAU'05
CSPN'05
DMIN'05
HCI'05
VISION'05
CSC'05
IKE'05
SAM'05
METMBS'05
AMCS'05
FUS'05
FECS'05
PDPTA'05
ICAI'05
SERP'05
ICOMP'05
CDES'05
ICWN'05
MSV'05
FCS'05
CISST'05
ISWS'05
PSC'05
MLMTA'05
CIC'05
ERSA'05
PLC'05
ESA'05
Editor: H. R. Arabnia
University of Georgia, GA, USA
Copyright by CSREA Press©
ISBN: 1-932415-48-3