Service aware Access Networks through Network Processor Technology

Koert Vlaeminck, Tim Stevens, Wim Van de Meerssche, Filip De Turck,
Bart Dhoedt, Piet Demeester

Abstract *This paper presents the design of a network system using network processor technology, aiming at introducing service awareness in access network nodes. Firewalling and network address translation were selected as reference applications, since they offer a clear and simplified use case of a more generic service enabler: packet classification. Our target platform is Intel's second generation network processor, more specifically the IXP2400. Implementation details of the service reveal some typical characteristics of network processor programming. A detailed performance analysis of the implemented services is presented and the system's bottleneck is identified, while possible solutions to alleviate this bottleneck are suggested.*

Keywords *access network, network processor, network services, service enablers, packet classification, firewall, network address translation.*

1 Introduction

Within the IST MUSE project [1] and other IST projects, such as NOBEL [3] and BREAD [2], substantial research effort is currently devoted to the realisation of the 'broadband for all' concept within Europe. Each of these projects focuses on different parts of the network (i.e., access, metro and core networks), working towards an integrated solution. The overall objective of MUSE is the development of a future, low-cost access and edge network, enabling the delivery of broadband multimedia services to subscribers.

Offering higher layer services in the access network involves increasing application awareness and intelligence in IP network nodes, while bringing IP awareness closer to the end-user. In order to meet subscriber demands, especially in residential access environments, the presence of additional packet processing units in network nodes is required. Due to their specific hardware architecture -based on far-reaching parallelism features- and their low power consumption, network processors are a perfect match for upgrading current networking equipment. We assume a (future) full Ethernet access network with IP-aware access multiplexers, as depicted in 1. This implies IP-awareness at the CPE, the DSLAM and the Edge Router (ER). 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).

This paper presents application architecture and performance details for a firewalling / network address (and port) translation (NAPT) service implemented on one of Intel's second generation network processors, the IXP2400. A firewalling / NAPT service is selected because it offers a clear and simplified use case of a more generic service enabler: packet classification. First, a pipelined architecture similar to Linux' Netfilter is mapped onto

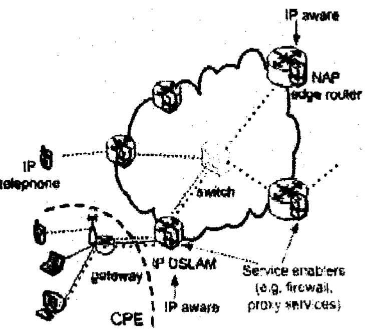


Figure 1: Depicts the assumed access network topology: a full Ethernet access network wi
aware access multiplexers (CPE: Customer Premises Equipment, DSLAM: DSL Access Multi
NAP: Network Access Provider)

the available hardware resources, revealing typical characteristics of network process
gramming. Later, the implemented solution is evaluated and compared to coarse an
models, identifying the system's bottleneck is processing power rather than memory l
which is reduced by the parallelism features. Possible optimizations are highlighted.

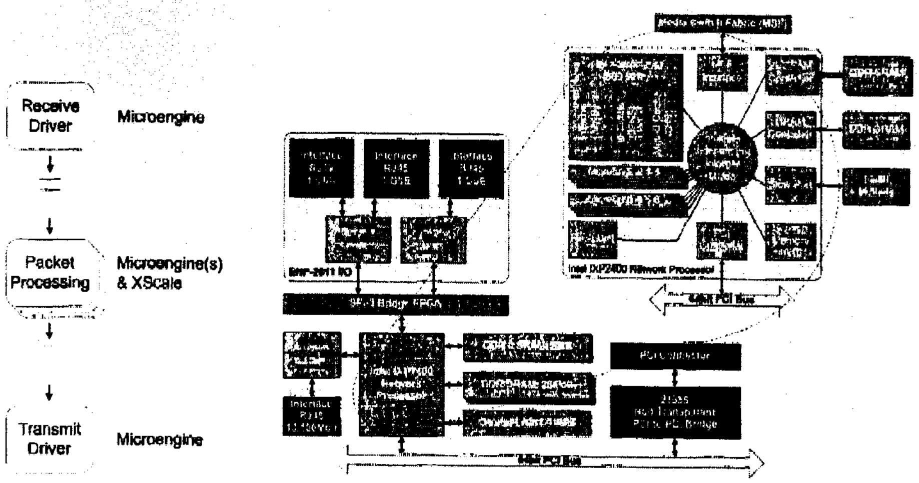
The remainder of this paper is structured as follows: Section 2 reports on relate
in this area. The target hardware platform, Intel's second generation network proc
presented in section 4, while the selected reference services, firewalling and NAPT, t
with their mapping to the selected hardware, are covered in section 4. Section 5 pr
detailed performance analysis, including coarse analytical models. Finally, Section 6
possible future work and concluding remarks.

2 Related work

Design and development of network systems using network processors (NPU)
topic. In August 2003, a special issue of IEEE Network was dedicated to the subject
articles were published providing an in-depth description of how specific data plane f
can be implemented using an NPU: one article on the implementation of a DiffS
router on a first generation Intel NPU [5], the IXP1200, the other on the implem
of an IPv4/IPv6 transition mechanism on that same NPU [6]. Another article in t
describes the design and implementation of an intelligent DSLAM using NPUs [7]
data plane functionality, again the IXP1200 was selected.

More recently, the October 2004 issue of IEEE Micro was dedicated to networ
sors. In this issue, we still see papers on the first generation Intel network processo
presents a simulation infrastructure for the IXP1200 which also provides an estimat
for measuring the simulated processor's power consumption. However, we also see t
generation Intel network processors appear: e.g. [9] presents the PRO3 hybrid N
tecture and compares it to the Intel IXP2400, while [10] compares, analyzes and
cryptographic algorithms on the Intel IXP2800.

Similar work on the implementation of service enablers on content processor
the Broadcom SB1250 (as opposed to network processors, like the aforementioned
families), was presented in [11].



(a) paradigm (b) enp-2611

Figure 2: Depicts the receive - process - transmit paradigm for programming an Intel IXP2xxx in 2(a) and the Radisys ENP-2611 Intel IXP2400 evaluation board in 2(b).

3 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 [13] depicted in Fig. 2(b), 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 [14], 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.

4 Programming an IXP2xxx

Intel second generation network processors are programmed using the receive - p transmit paradigm [15] as depicted in Fig. 2(a): receive, process and transmit tasks different processing units (microengines and / or XScale core) with queues betweel The microengines can run a series of sequential processing tasks (pipelined approa pool of parallel processing tasks or a mixture of both.

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

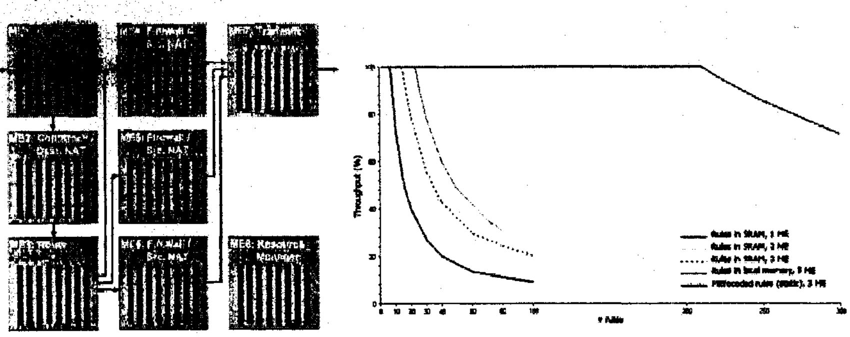
Furthermore, firewalling / NAPT service offers a clear and simplified use case generic service enabler: packet classification. Performance evaluation of a firewall implementation on Intel network processor hardware gives a good indication of h network services will perform on this architecture, since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

We based our implementation on the netfilter / iptables [16] forwarding chain and iptables are building blocks of a framework that enables packet filtering, netwo (and port) translation and other packet mangling inside the Linux kernel. Input a chains process packets destined to and originating from a host. As our firewal service is meant for integration in access network equipment, only the forwardir important for data plane traffic.

The netfilter forwarding chain consists of several hooks, which identify the packet processing tasks: (i) First, the pre-routing hook performs connection tr destination address / port translation; (ii) then the routing decision is made; forwarding hook it is decided whether the packet is to be forwarded or not decision); (iv) finally, the post-routing hook performs source address / port tran

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

Mapping the netfilter components to the IXP2400's processing units is relativ forward, as depicted in figure 3(a). Receive and transmit code each occupy one r A third microengine is used for resource management, on which 6 threads are acti ory allocation, (ii) free memory, (iii) search for conntracks that timed out, (iv) TCP ports for NAPT, (v) allocation of UDP ports for NAPT and (vi) allocat ID's for NAPT. This leaves five microengines for packet processing: the pre- occupies one, routing occupies a second and firewall & post-routing can be co third. Earlier experience [17] with Intel's first generation network processor (t has learned that the sequential search through the firewall rules is very reso and induces a performance bottleneck. A first implementation, where packets against the firewall rules at the core, only allowed 1500 new connections to



(a) firewall / NAPT (b) performance

Figure 3: Depicts the mapping of the firewall components on IXP2400 hardware in 3(a), while 3(b) depicts the performance of this implementation for an increasing number of rules, measured using two GbE interfaces (one for upstream, one for downstream) and minimum sized packets. All 8 threads per ME are active.

second on IXP1200 hardware[1]. Implementing the search through the firewall rules at the microengines increased this number to up to 58000 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.

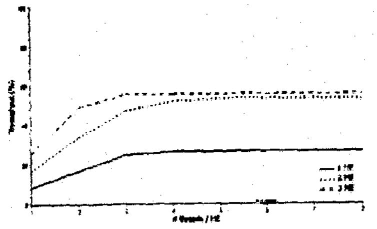
5 Network Processor performance evaluation

A Spirent Smartbits 6000 network performance analysis system [18] 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.

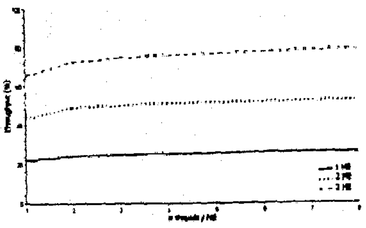
As already stated in the previous section, the sequential search through the firewall rules limits the IXP's performance. In stateful firewalls however, packets of known connections can be immediately accepted or rejected, based on connection tracking information, and only the first packet of each connection 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. 3(b) and will be explained below.

With only one firewalling microengine, our IXP2400 evaluation board is able to handle six to seven firewall rules at gigabit speed. This is clearly insufficient. Duplicating the

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



(a) rules in SRAM



(b) rules in local memory

Figure 4: Depicts the throughput of the IXP2400 firewall / NAT implementation when processin 30 rules for a varying number of threads per ME, with one to three MEs running the packe classification code.

firewall code on a second microengine doubles this performance: up to 14 firewall rules ca be handled at gigabit speed. This proves the sequential search to the firewall rules is indee a bottleneck to the IXP's performance. Adding a third firewalling microengine, however, r longer increases performance. The bottleneck has shifted from processing the firewall rul to fetching those rules from SRAM memory, as we will show below (cf. Fig. 4(a)).

Although a multithreaded design – remember that each IXP2400 ME has support f eight hardware threads – helps in hiding the latency of the SRAM accesses, required f fetching the firewall rules from memory, faster memory could save some cycles per firew rule when processing a packet. We made an implementation where the rule set is stor in the firewall microengines' local memory. Data can be read from this memory in or three cycles (cf. 90 cycles for SRAM access). This implementation, able to handle sev to eight firewall rules per microengine at gigabit speed, is actually only slightly faster th the implementation where rules are stored in SRAM. This shows the multithreaded des is actually very efficient at hiding memory latency.

The local memory implementation does, however, have another advantage: each mic engine has its own local memory. Adding a third firewalling microengine to this implem tation further increases performance: up to 23 rules per packet can be processed at gig speed, using three MEs. On the downside, we were only able to store up to 80 rules in limited – 640 longwords – amount of local memory.

Fig. 4 illustrates the effect of multithreading on both implementations. The IXP2400 a 30-rule firewall – remember that none of the presented implementations can process n than 23 rules at linerate – running on 1 to 3 microengines for a varying number of thre Fig. 4(a) depicts the throughput of the SRAM implementation, while Fig. 4(b) depicts throughput of the local memory implementation.

Again we see that adding a second firewall ME doubles performance for the SRAM in mentation, while a three ME implementation does not further increase overall througl The performance of the 3 ME SRAM implementation saturates at 4 active threads pe croengine, while the 2 ME SRAM implementation requires 6 active threads per microe before reaching maximum performance. This indicates that the SRAM memory is at sustain up to 12 firewall threads before becoming a bottleneck.

For the local memory implementation we see that enabling additional threads doe influence performance that much. From 2 threads per ME on, performance is roughly stant. This result was expected: since accessing a longword from local memory only 3 clock cycles, multithreading is no longer beneficial for hiding memory latency. I threading [illegible] positive influence, however, since the packet headers still have to be fetched from [illegible]. From Fig. 4 it is also clear that the one and two ME local memory implementations have similar performance as the SRAM implementations on one and two MEs respectively. Adding a third ME to the local memory implementation further increases performance however, which is not the case in the SRAM implementation.

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: using 3MEs, up to 210 rules per packet could be handled at gigabit speed, as is illustrated in Fig. 3(b). 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.

6 Conclusion and Future Work

In this paper we evaluated the suitability and performance of network processors, aiming at increasing application awareness and intelligence in IP network nodes. We chose packet classification as a service enabler (evaluated through a firewall / NAPT implementation) and Intel's IXP2400 as target platform. Typical characteristics of network processor programming – i.e. receive - process - transmit paradigm – were revealed. A detailed performance analysis 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 [19] look very promising and will be reported upon in a future publication.

Acknowledgement

This work is supported by the IST FP6 MUSE project [1].

Contact information

Koert Vlaeminck (`koert.vlaeminck@intec.ugent.be`)
Department of Information Technology (INTEC)
Ghent University - IBBT - IMEC
Sint-Pietersnieuwstraat 41, B-9000 Gent, Belgium
Tel. +32 (0)9 331 4942, Fax. +32 (0)9 331 4899

References

[1] *Multi Service Access Everywhere*, `http://www.ist-muse.org/`.

[2] *Broadband in Europe for All: a Multi-Disciplinary Approach*, `http://www.ist-bread.`

[3] *Next Generation Optical Networks for Broadband European Leadership*, `http:/ ist-nobel.org/`.

[4] H. Vin, R. Yavatkar, *Guest Editorial: Network Processors*, IEEE Network, Jul/Aug 200 17, No. 4, Pages 10-11.

[5] Y.-D. Lin, Y.-N. Lin, S.-C. Yang, Y.-S. Lin, *DiffServ Edge Routers over Network Proc Implementation and Evaluation*, IEEE Network, Jul/Aug 2003, Vol. 17, No. 4, Pages 2

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

[7] R. Neogi, K. Lee, K. Panesar, J. Zhou, *Design and Performance of a Network-Processo Intelligent DSLAM*, IEEE Network, Jul/Aug 2003, Vol. 17, No. 4, Pages 56-62.

[8] L. Yan, J. Yang, L. N. Bhuyan, L. Zhao, *NePSim: A Network Processor Simulato Power Evaluation Framework*, IEEE Micro, Sep/Oct 2004, Vol. 4, No. 5, Pages 34-44.

[9] I. Papaefstathiou, e. a. , *PRO3: A Hybrid NPU Architecture*, IEEE Micro, Sep/Oct 20 4, No. 5, Pages 20-33.

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

[11] T. Vermeiren, E. Borghs, B. Haaodorens, *Evaluation of software techniques for parall processing on multi-core processors*, IEEE CCNC 2004, Las Vegas, Jan 2004.

[12] L. Zier, W. Fischer, F. Brockners, *Ethernet-Based Public Communication Services: C and Opportunity*, IEEE Communications Magazine, Mar 2004, Vol. 42, No. 3, Pages

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

[14] *The ARM Instruction Set Architecture*, `http://www.arm.com/product architecture.html`.

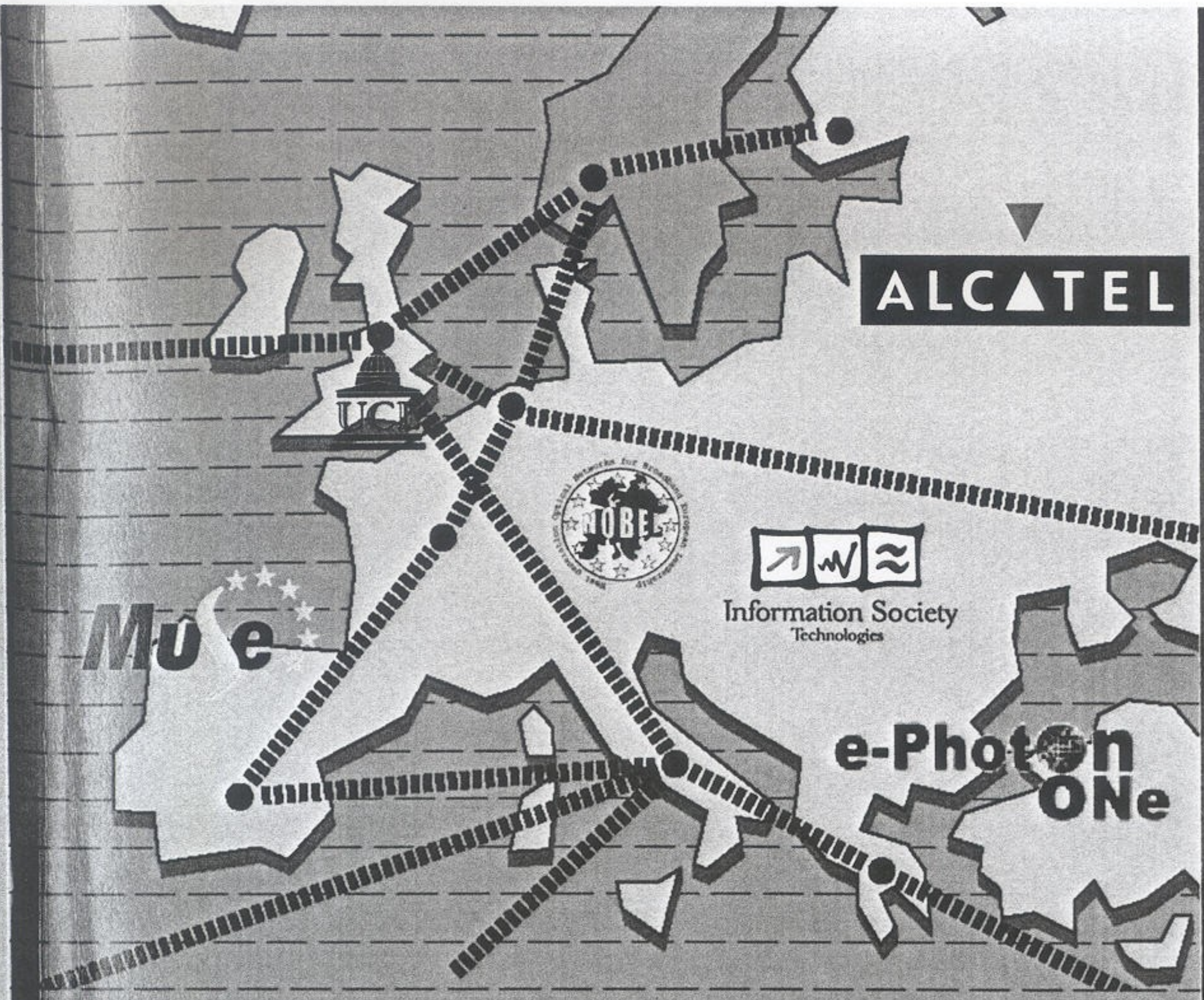
[15] E. J. Johnson, A. R. Kunze, *IXP2400/2800 Programming, The Complete Microengin Guide*, Intel Press, 2003.

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

[17] K. Vlaeminck, T. Stevens, F. De Turck, B. Dhoedt, P. Demeester, *Deployment o processors in access networks to provide service enabling functions: evaluation re* European Conference on Networks & Optical Communications (NOC2004), Jun 29-J Proceedings, Pages 70-77.

[18] *Spirent Smartbits 6000 Product Overview*, `http://www.spirentcom.com/document`

[19] S. Hazelhurst, A. Attar, R. Sinnappan, *Algorithms for Improving the Dependability and Filter Rule Lists*, International Conference on Dependable Systems and Netw 2000), Jun 25-28, 2000.



NOC 2005

10th European Conference on
Networks and Optical Communications

University College London, London, UK
July 5th- 7th, 2005

Edited by
J.E. Mitchell and D.W. Faulkner

ISBN: 0-9538863-8-7
Published by the Communications Engineering
Doctorate Centre, UCL