Architecture and Applications of PLATO, a Reconfigurable Active Network Platform

Apostolos Dollas, Dionisios Pnevmatikatos[1], *Nikolaos Aslanides, Stamatios Kavvadias, Euripides Sotiriades, Sotirios Zogopoulos , Kyprianos Papademetriou*

Department of Electronic and Computer Engineering
Technical University of Crete
Chania, 73100
Greece

Contact e-mail: dollas@mhl.tuc.gr

Nikolaos Chrysos, Konstantinos Harteros

Institute of Computer Science (ICS) – FORTH
Heraklion, 71110
Greece

Emanouil Antonidakis, Nikolaos Petrakis

Department of Electronics
Technological and Educational Institute of Crete
Chania, 73100
Greece

Abstract

A new, configurable architecture has been designed and built in order to serve as a platform for experimentation with active networks. This architecture, named PLATO, provides 4 physical bi-directional connections for ATM networks, large reconfigurable resources, 256 Mbytes SDRAM for buffer space, a PCI port, and auxiliary expansion ports. Several applications are presented for this platform, one of which has been prototyped on the PCI Pamette and on PLATO. Detailed simulations and experimental results show that, for some applications, a significant improvement can be obtained using this approach as compared to using conventional network architectures.

KEYWORDS: Active networks, reconfigurable computing, protocol boosting.

1. Introduction

The term "active networks" is the notion for smart networks. It principally refers to extending the functionality of networks to support useful computation on the data sent over them, but it also includes the increase of network flexibility, adaptability, security and functionality in general. The typical flexibility offered by active networks is automatic, rapid protocol deployment, making networks programmable. This in turn, leads to the necessity of processing power on active network nodes. Conventional networks provide routing of cells based on some information found in the header of each cell. By comparison, in addition to the above functions, active networks either use the payload of the cell for routing purposes (e.g. for dynamic balancing of loads in networks), or process the payload of the cell as it goes through the network (e.g. for data extraction).

With very few exceptions, noted in section 2, below, almost all active network projects are software based, due to the (correct) assumption that networks of the future will contain processing power comparable to high-end processors of today. Whereas this approach is excellent for the exploitation of new ideas, it leaves out interesting areas such as real-time payload processing of cells, which may be needed for important applications. Such applications can be the detection of Denial-of-Service (DoS) attacks, real-time load balancing for e-commerce servers, real-time network based speech recognition servers for v-commerce, protocol boosting, etc. To experiment with such applications we have developed PLATO, a reconfigurable platform for ATM networks. PLATO has been implemented and is undergoing tests, whereas the PCI Pamette[1] has been used as a rapid prototyping tool, to test several subsystems and applications. Although network applications with FPGA's have been implemented in the past [2][3], the vast range of real-time active network applications that can be developed with last generation, large FPGA's makes this project promising.

Section 2 presents related work, section 3 has the architecture of the new platform, section 4 presents examples of specific applications for the new platform, section 5 presents performance results from simulations and from prototyping with the PCI Pamette, section 6 has the present status of the project, and section 7 has conclusions and future work, followed by the acknowledgements and bibliography.

[1] Also at ICS-FORTH

2. Related Work

In order to investigate the design space for an active network node's architecture, research has spread between two discrete approaches: in-band and out-of-band node programming. The in-band or integrated approach uses cells containing both program and data, which are called *capsules*. When a capsule arrives at a network node, the program that is contained in the capsule is executed using the data in the capsule. In the out-of-band or discrete approach, code implementing a service is loaded on the network node(s), and passing traffic can then utilize the service. In this last case, a process of upgrading the node's capabilities is needed, in contrast to the dynamic nature of the integrated approach. On the other hand, the need to efficiently address the wide range of possible active network applications, and the indispensability of security and bandwidth considerations, seem to lead to a convergence of the two extremes mentioned above.

As we move towards less dynamic network programming models, it becomes easier to capture hardware implementations, without the need for extensive operating system and language support on the active node. Towards the opposite direction, we find increased flexibility in network application programming and service deployment, at some potential cost in efficiency since extra latencies are introduced by the software approach taken.

The ANTS project at MIT [4][5] introduced the notion of capsules. An ANTS capsule integrates a cell's data and a reference to a Java forwarding routine, which must be loaded (if not present) on the active node before the capsule can be processed. ANTS runtime must be installed on active nodes, to do the processing introduced by the forwarding routine and also handle security issues through techniques like sandboxing and Java bytecode verification. Smart Packets at BBN [6], also uses capsules to extend the diagnostic functionality in the network, using a specially developed language (Spocket) and run-time environment. The SwitchWare project at the University of Pennsylvania and Bellcore [7] relies on active packets, carrying limited functionality programs written in PLAN [8], resident or loaded out-of-band active extensions to provide additional functionality and an active router infrastructure. Verification, authentication and secure bootstrap techniques are applied to insure integrity, safety and security.

In the CANES project at Georgia Tech and the University of Kentucky [9][10], the functionality of an active network node is divided between the Node Operating System and Execution Environments running on top of it, the latter of which correspond to composition methods for defining new services (e.g. ANTS, PLAN).

Open signaling (Opensig), refers to an instance of programmable networks that clearly separates network control from information transport [11]. The Xbind [12] broadband kernel abstracts node resources and supports the programmability of the management and control planes in ATM networks, based on CORBA [13] middleware to incorporate multiple vendor switches. The Genesis project [14] at Columbia, implements the Virtuosity kernel, which automates the spawning of network architectures in a process of *profiling*, *spawning* and *management*. *Profiling* captures a "blueprint" of the architecture. *Spawning* sets up the virtual network's topology and address space and binds transport control and management objects. *Management* supports virtual network architecture refinement and resource management.

To our knowledge, only two projects have considered hardware platforms for active networks. The ANN project [15][16] at the Washington University in St. Louis devised a hardware implementation for a scalable active network node architecture for gigabit environments. Their platform consists of a general purpose CPU, one or two FPGA's, 64MB of memory and an ATM Port Interconnect Controller (APIC) chip connected on a PCI bus. In addition, their Router Plugins [17] research software platform is used as a framework for the development of a NodeOS. The Programmable Protocol Processing Pipeline (P4) project [18][19], has implemented a platform comprised of a set of ALTERA FLEX8000 FPGAs acting as processing elements (PEs) and a switching array selecting which devices are engaged in processing. The PEs are arranged in a pipeline exchanging data through FIFOs and FPGAs can be moved in and out of the pipeline to be reconfigured. This testbed has been used to implement forward error correction (FEC) protocol processing (boosting), on a single OC-3 ATM link.

Active network applications include active reliable multicast [5], self-organizing network caching [10], multicast video distribution [15], online auctions [4], active bridging [7], protocol boosters [19], active congestion control, distributed firewalls, and packet filtering. Software approaches to such services (e.g. ANTS, Smart Packets, PLAN and CANES), have historically been implemented over IP networks. In the context of ATM networks, even though QoS features are inherent, signaling complicates service creation. Our platform, introduced in the next section, has the potential to explore the capabilities of an active, 155 Mbps per link, 4x4 ATM switch.

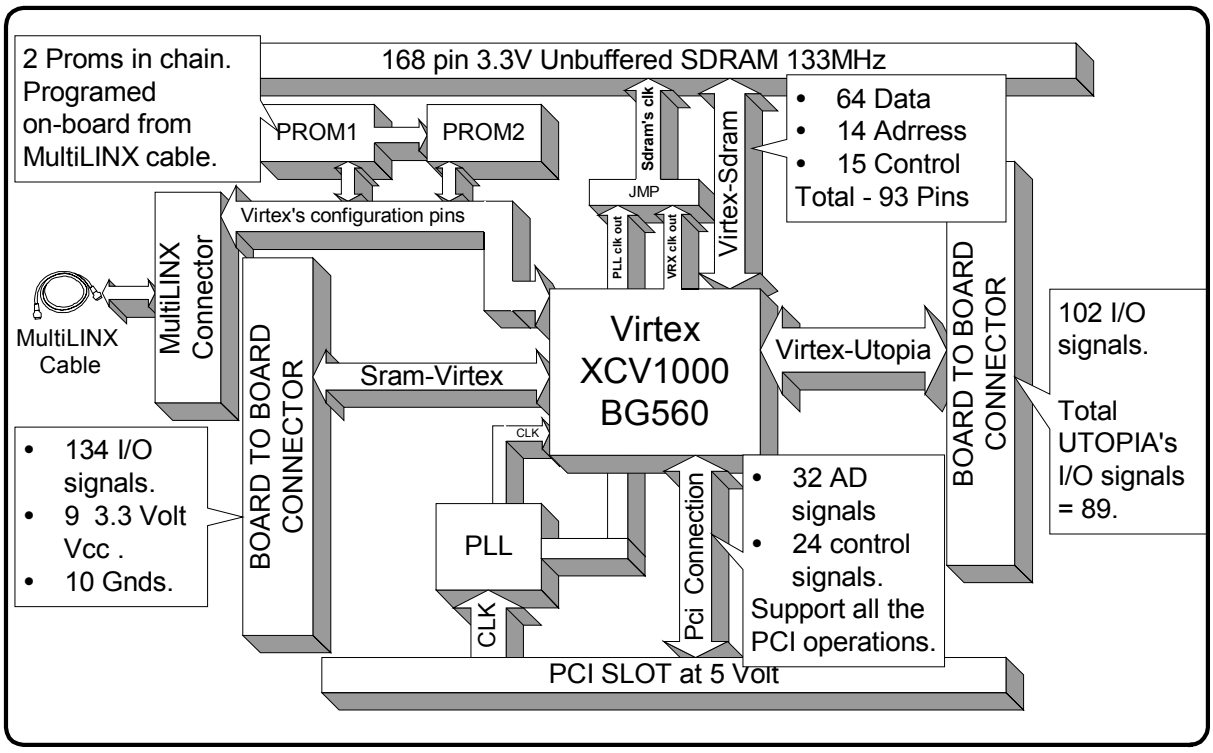

Figure 1. Architecture and Block Diagram of PLATO

3. PLATO Architecture

The architecture of PLATO will be presented together with one particular design, as shown in Figure 1. This is deliberate, because the platform is mainly aimed to be a tool for experimentation and design library development, and hence was derived from a minimum set of functionality requirements, as follows:

- Ability to be connected to fiber or copper ATM networks, ability to be connected in the future with Ethernet (10/100 or Gigabit)
- Minimal delay in hardware processing of the cells, and ability to implement protocols with no need to communicate with the host processor
- On-board buffer space for streams of cells, on the payload of which, processing will be performed
- Ability to communicate with a general purpose computer, for further processing, downloading of statistics, or partial reconfiguration of the FPGA
- Extra connectors for expansion

The PLATO platform has a large FPGA (versions with Xilinx Virtex XCV 1000 and ALTERA 20K400 have been designed), which in addition to the clock generation circuit, programming ports, etc. has four main ports:

- A UTOPIA level 2 port, to provide the physical connection to daughterboards with copper or optical fiber outputs – the daughterboard also has the UTOPIA level 2 - ATM framing circuitry
- A 256 MB 133MHz SDRAM port for buffer space
- An auxiliary port which will be used in another project [20] for SRAM look up tables
- A PCI bus port for communication with the host.

Although the general topology resembles several existing products, the need for new hardware was largely mandated by the numbers of pins needed for each port, as well as the voltages in each case (e.g. PCI at 5V, SDRAM at 3.3V, etc.).

PLATO has several similarities with that of the Washington University ANN active network project, as well as with general-purpose FPGA platforms, such as the SLAAC. A notable difference with ANN is that PLATO does not have a general-purpose processor embedded in it. Although we recognize the usefulness of such a processor, the goal of PLATO was to experiment with the performance that the reconfigurable resources offer in several applications, and, the development of VHDL design libraries for operations such as cell disassembly and reassembly, payload extraction, etc.

The general architecture is independent of the FPGA vendor, and indeed we have proceeded with the design of two versions, one with Xilinx Virtex and one with ALTERA 20K400 FPGA's. This way we hope to get results and gain insight on how the architecture of the FPGA's and the corresponding CAD tools affect the system level performance. As the Xilinx version of the architecture has already been implemented, the remainder of this paper will focus on this version.

4. Applications

Several applications are under development for PLATO, including the implementation of a real-time continuous speech recognition server [20] for ATM networks, and experimental work on Denial of Service (DoS) attack detection. In this section we will present two applications: an active 4x4 ATM switch with simple processing capabilities, and a protocol boosting application. The first application has already been prototyped in hardware using the PCI Pamette, and subsequently ported on PLATO. Therefore we have performance results to report even before our platform is fully functional. The second application gives a clearer picture of the kind of processing that can be performed by our system.

4.1 Active 4x4 ATM Switch

Recognizing that most every active network application for PLATO will involve cell disassembly and reassembly, cell routing, and the ability to perform some processing (however sophisticated) on the header *and the payload*, the first application is a 4x4 switch with the ability to screen some types of cells. These are initially "ping" cells because they are short and they are associated with some forms of DoS attacks. This switch will form the foundation for subsequent applications, as it implements all the basic and necessary functions that a common ATM switch does. The switch is divided in three parts: the input Header Processor, the multiplexing and application interface (which interfaces with the hardware that makes the network "active") and the Output Header Processor.

When the UTOPIA level 2 interface of the switch receives an ATM cell, it places it in a FIFO (there is one FIFO per incoming link). The switch starts to process the cell by disassembling it and loading the respective registers with all the fields of the header (VPI, VCI, PT, CLP, and HEC). When the VPI and VCI of a cell have been received, the ATM switch evaluates, according to an internal look up table, the output VPI, VCI and physical link. The ATM switch has four internal look up tables, one per incoming link. Each look up table is memory based with 256 positions each. This way the switch can handle cells in all four input ports simultaneously. The size of the tables is small for an ATM switch, but it is big enough for experiments on active network applications. As the header is processed, data from the payload of the cell continues to be received from the UTOPIA level 2 interface for six system cycles. Since the payload may determine whether a cell will be rejected or not, and until conflict handling circuitry determines availability of the necessary output link, all the data are stored in an input FIFO. The number of the positions that the FIFO should have is also six, but depending on the technology that is used (Xilinx or ALTERA) the number of positions can be either six or fifteen.

On the multiplexing and application interface, the control circuitry checks if there are any collisions, and gives permission or rejection to every cell. This part also gathers all the signals from every input and output port of the switch, and performs the actual switching of cells. Internal input and output port busses are 48 bits wide. This is the number of the header bits plus the number of the bits of the first payload word. The priority of a cell to get output permission is based on the time of its arrival. If two or more cells are received on the same cycle from different input ports and they need the same output port for transmission, then the cell that came from the lowest numbered input port passes and the others are rejected. This arbitrary algorithm was used for reasons of rapid prototyping and is not inherent to the architecture. Due to the wide bus that is connected to the multiplexing unit and the open architecture of the control of this unit, this is where processing of cell payload can take place, by interfacing additional application modules to it. For example, assuming that the cells of the application have a specific VPI and VCI, the switch can easily detect them and regard the application as a fifth input/output port of the switch. Applications on the PLATO architecture can have access to 256 MB of SDRAM and can process single cells, or up to a large number of cells, or even keep multiple copies of the cells. As the switch regards every application as another input/ output port, more than one application can be connected to the switch in parallel.

The third part of the switch is the output header processor. When a cell gets permission to pass to an outgoing link, the output processor starts to reassemble the header of the cell. It reads the 48-bit bus, loads the respective output registers and then initiates transmission to the UTOPIA level 2 interface.

4.2 Wormhole IP over ATM Routing Filter

Fast network access and throughput is of capital importance in today's Internet. Meanwhile, IP is the uncontested protocol of choice for data communications. ATM technology finds widespread use, mostly because of its use of (relatively) small, fixed-size "cells" as the transfer unit. The use of small cells allows for high-speed hardware switching, while quality-of-service (QoS) is possible through the use of preemption of cells in different service classes. Several techniques have been proposed and are being used for routing IP packets over ATM networks: LAN emulation, classical IP over ATM, ATM under IP [21], etc. Generally, these software methods have a number of costs, overheads and associated delays: (i) a very large number of VP/VC labels are consumed, or (ii) until a flow has been recognized, the first few cells in the flow are not switched; and (iii) new connections for new flows are established in software (e.g. ATM signaling), thus suffering long delays.

Wormhole IP over ATM (WIPoATM) [22] is a solution that combines the following characteristics: (i) it performs full IP routing per IP packet, (ii) it overlaps (in a pipelined fashion similar to the wormhole routing in computer networks) the transmission of IP packets over different VP/VCs, (iii) it does so without suffering the initial connection set-up delays. WIPoATM is a routing filter that sits in the entry point(s) of an ATM network exclusively for the support of IP traffic. When the first cell of an IP packet is received, WIPoATM performs the necessary IP routing table lookup using a fast IP routing lookup [23] implemented in hardware, and allocates a (pre-established) VP/VC for the transmission of this IP packet. Subsequent cells of this packet will use the chosen VP/VC. The process is repeated for the next IP packet choosing a possibly different VP/VC. By managing dynamically a number of VP/VCs, WIPoATM can avoid the (high) latency of establishing connections using ATM signaling. The rest of the ATM network knows nothing of IP packets, just receives and switches the cells as received from the WIPoATM routing filter. This way, the hardware can balance the IP traffic and manage network resources (VP/VC) and the rest of the network does not know anything about IP packets. On the other end a second filter recomposes the ATM cells so that they arrive at the recipient from the known VP/VCs

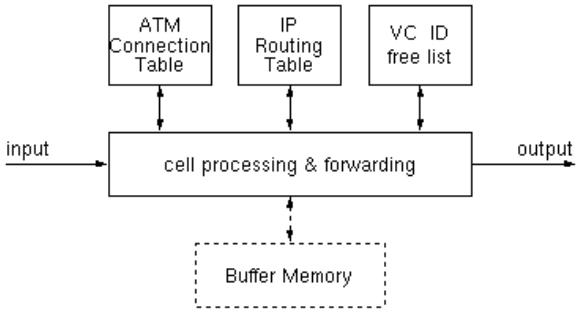

Figure 2. Functional Block Diagram of the WIPoATM Routing Filter

and be accepted as legitimate IP over ATM traffic. The proposed Wormhole IP routing filter has a number of advantages:

- it works together with standard, existing ATM equipment and allows the seamless integration of both IP and native ATM traffic in the same networks;
- the quality of service of native ATM traffic can stay unaffected by the added IP traffic, while IP can benefit from ATM's QoS capabilities;
- cell routing delay is minimized owing to virtual-cut-through routing --segmentation and reassembly delays at *intermediate* routers are eliminated;
- the number of pre-established connections (labels) is small and fixed, and does not grow with the size of the total network (as in tag switching), yet all cells are routed through pre-established connections.

4.2.1 Organization and Operation of the Wormhole IP over ATM Routing Filter

The organization of a Wormhole IP over ATM routing filter is shown in Figure 2, and consists of the ATM connection table, the IP Routing table, the VC ID free list and an optional buffer memory. The filter operates by looking at each ATM cell that passes through it. For each incoming cell, we look up its virtual path (VP) and virtual circuit (VC) identifiers in the filter's ATM Connection Table. If the cell belongs to a native-ATM connection, it is forwarded to the output, possibly after undergoing VP/VC translation. Otherwise, the cell belongs to IP traffic and we distinguish three possible cases:

1. This is the first cell of an IP packet. The cell contains the IP destination address (either without or with multi-protocol encapsulation) that must be looked up in the IP Routing Table. The routing table specifies the desired output VP; an unused VC identifier in that VP is then requested from the VC ID free-list. If no such unused VC currently exists,

then the cells of this packet must be buffered in expectation of some VC getting freed, or the cell can be dropped.

2. This is a middle cell of an IP packet. The ATM Connection Table contains all the necessary information for the VP/VC translation, and the cell is forwarded to the ATM network without further processing.
3. This is the last cell of an IP packet (indicated by the PTI field of the cell header according to the AAL5 segmentation). The temporary, wormhole connection that had been established for this IP packet has to be torn down; this is accomplished simply by marking its state as inactive in the connection table, and by returning the output VP/VC identifier to the free list.

4.2.2 Mapping the Wormhole IP over ATM Routing Filter on PLATO

WIPoATM fits very nicely in the PLATO architecture. The ATM Connection Table, and the IP Routing Table can be stored in the SDRAM module(s) while the VC ID list can be kept either in memory inside the processing FPGA, or in the SDRAM. While PLATO offers up to 4 bi-directional 155Mbps links for input and output, WIPoATM requires 2 bi-directional links (as it is a bi-directional filter). The remaining 2 ports can be used to implement a second WIPoATM filter on the same prototype, or to double the bandwidth of a single WIPoATM Routing Filter. The amount of logic offered by PLATO is more than enough for the implementation of two such filters.

5. Performance

There are several published results from projects similar to PLATO (e.g. ANN), with a hardware approach to active networks, as well as results from software approaches, in which ATM or 100Mbps Ethernet data are sent via a network interface card to a CPU for processing. Although, as expected, some projects are complete and their results are based on older technology (e.g. PLAN) and some projects are just now producing results and these are not fully optimized yet (e.g. ANN, PLATO), the similarity of applications in several of these projects allows for conclusions to be drawn in terms of general characteristics, rather than specific numbers. The goal of this section is to demonstrate that reconfigurable hardware offers real and tangible advantages over software approaches in active networks, for some types of applications. Such advantages may stem from the elimination of the protocol processing, or from the elimination of the operating system overhead required to pass the cell payload information to an application. In either case, to the extent that the benefit is real, it is worth reporting, even as an indication of what functions can be performed in hardware in the future.The overall latencies we are interested in take into account the cut-through latency from the time a cell enters the switch from the UTOPIA interface, until the time the cell exits the switch towards the UTOPIA interface. No adjustments have been made for implementation inefficiencies of the 4x4 switch in PLATO vs. highly optimized commercial switches. The PLATO results come from two sources: simulated operation of the switch at 60MHz on a Xilinx Virtex, and, actual operation of the switch at 33MHz on the PCI Pamette. Because to date simulation frequencies have been found to be conservative, the 60MHz Virtex results are used, and we consider the Pamette results to be useful as functional verification of the system and not indicative of its performance bounds. Although we do have results from the Virtex-based PLATO, these are (at present) on functional correctness only.

5.1 PLATO Performance

In order to measure the PLATO latency, a simple active network application has been designed. This application "cuts" every cell that performs an ICMP ECHO (i.e. in Unix terms a "ping" or a "ping reply"), and passes through the switch all other cells. This application will be compared to similar operations in other active networks, including the processing of a zero payload capsule, IP forwarding, and ICMP ECHO.

The PLATO latency in cycles that a cell needs to pass through the switch is:

Cycles for cell switching = Cycles for Header disassembly + Cycles for Look up table access + Cycles for Rejection/Pass decision + Cycles for header Reassembly =4 + 1 +2 + 1 = **8 Cycles.**

The total number of cycles, including the UTOPIA interface is:

Total Cycles = Cycles to receive the cell from UTOPIA + cycles for cell switching + cycles to transmit the cell to UTOPIA = 29 + 8 + 28 = **65 Cycles**

Thus, 65 cycles are needed for PLATO, including cell disassembly and reassembly. For the ICMP ECHO (ping) screening, detection, drop of ECHO packets, and pass of non-ECHO packets, an additional **27 cycles** are needed, raising the total cycles to **92 cycles**.

The net results (Virtex, simulated, with post place and route indicated frequency of 60MHz) are:

$$\textbf{PLATO Latency (switch)} = \frac{\textbf{65 cycles}}{\textbf{60 MHz}} = \textbf{1.08 µs}$$

$$\textbf{PLATO Latency (switch + ECHO app.)} = \frac{\textbf{92 cycles}}{\textbf{60 MHz}} = \textbf{1.53 µs}$$

5.2 Network Interface Performance

Some older results from our group [24] as well as from other groups [25][26] measure the latencies of a network, including processor overhead and full protocol implementation. We include these results for completeness, because the detection of a simple ECHO at the processor level incurs the network interface overhead, as well as the appropriate protocol processing, operating system, and application overheads.

Performance measurements were made in [24] with a LAN emulation device driver using an 133MHz Pentium processor running Windows NT 4.0. The Network Interface is IDT77901 ATM (NICStAR controller), plugged on a PCI bus running at 33MHz. A socket-based application was developed for the experiments. The NICStAR controller uses DMA on the host memory to reassemble CS-PDUs from the ATM cell payloads which are received from the network. When transmitting, as CS-PDUs become ready, they are queued in host memory and segmented by the NICStAR into ATM cell payloads. Software Echo Latency (SEL) is the time needed to transfer the data to the host memory, collected by the application and then transmitted to the PCI bus (read back operation). SEL numbers have very small variations for messages up to 100 bytes. The numbers presented below correspond to a 1 byte message transmitted to/received from the application. The times, below, were measured with a logic analyzer. The total time to read a cell from a physical link and write it back to the physical link is:

Time_IDT = Reassembly Lat.+ PCI BAL (NIC Initiator) + SEL + PCI BAL (NIC Target) + Segmentation Lat.
For UDP: 3.66 + 0.15 + 407.1 + 0.39 + 27.07 = **438.37 µs**

Performance measurements in [25] have been done using a "pure" device driver, allowing to send/receive cells via the kernel. The experimental set-up consists of a 60 MHz SPARCstation-20 running SunOS 4.1. The Network Interface is FORE Systems SBA-100 ATM (No co-processor on the NI) located on the Sbus (32-bit bus running at 20 or 25 MHz). The only function performed in hardware beyond serializing cells onto the fiber is ATM header CRC calculation. The card does *not* calculate the required AAL5 CRC. No DMA or segmentation and reassembly of multi-cell packets are supported by the interface. The segmentation/reassembly of the cells and the send/receive operation for individual ATM cells (controlled by the device driver) are located on the kernel. The benchmarks show that the timings for a single cell are:

Write system call: 28.2 µs
Read system call: 34.1 µs

Time_FORE_SBA = 34.1 + 28.2 = **62.3 µs**

The U-Net architecture developed at Cornell University [26] is an application which sends/receives cells directly via MUX to/from a Network Interface. The kernel is only involved in connection to set-up/shut-down. The experiments have been done with a Pentium 133MHz workstation, running the Linux operating system. The network interface is the FORE Systems PCA-200 that *includes* an on-board processor which performs the segmentation and reassembly of cells as well as transfers data to/from host memory using DMA. The PCA-200 consists of a 25 MHz Intel i960 processor, 256Kbytes of memory, a DMA-capable PCI-bus interface at 33MHz, a simple FIFO interface to the ATM fiber and an AAL5 CRC generator/checker. The U-Net implementation uses custom firmware to implement the U-Net architecture directly on the PCA-200 NI. The total send/receive operation time is:

Time_UNET = 11.5 + 13 = **24.5µs**.

We observe that the 438.37 µs of the IDT results include full protocol overhead for UDP/IP over ATM (which is slightly faster than TCP/IP over ATM). The 62.3 µs of the earlier Cornell measurements protocol-wise include only the ATM and AAL5. Lastly, the 24.5 µs of the UNET measurements are obtained by bypassing the kernel, and using a processor on the network card. Even accounting for newer technologies (for which we do not have published measurements) we notice that there is substantial overhead to reach the application at the processor from that network card.

5.3 Active Networks Performance

The ANTS experiments [28] have been done on a SUN Ultra 1 running Solaris 2.6, and with a 100Mbps Ethernet network interface. For a zero-payload capsule with built-in Java run-time facilities, the overall send/receive operation takes roughly **525µs**. This time includes IP processing for the forwarding of the capsule.

The PAN project [29] is implemented on a 200MHz P-Pro with 64MB RAM, running Linux, having a 100Mbps Fast Ethernet Network Interface. For an ICMP ECHO

experiment which includes IP forwarding in the intermediate node, which is essentially identical to our application, using the active mode of the network, the latency is 70 μs to process an ICMP ECHO by the kernel, 133 μs for the same operation from user space in x86 code, and 448 μs for the same operation in Java code.

The PLANet project [30] at the University of Pennsylvania is an active network architecture implementation. It uses the PLAN language for programs within packets of the network, which are subsequently executed in the network nodes. The hardware comprises of two 300MHz Pentium II processors running Linux, and the network interfaces are 100Mbps Ethernet cards. The PLANet application we will consider is an ICMP-based ping program (ping application using the PLAN language), and its latency is 100 μs when running in user space.

The ANN project [31] comprises of several experiment configurations, e.g. with a 233MHz Ppro host processor having an 155Mbps ATM interface, and several versions of the Washington University hardware platform (cards with one or two FPGA's plus embedded 166MHz Pentium MMX processor). The main source for results from the ANN project is their URL [31]. Several software experiments have been conducted, and we will include in this comparison the results for IP forwarding, which do not come from the ANN hardware card. The latency is 84.5 μs for a router plug-in architecture implemented in the kernel. We have not seen experimental results from the ANN hardware implementation, but we expect that these would be similar to ours.

5.4 Overall Comparisons

The basic characteristic of PLATO is that it connects directly the Network Interface with the application, bypassing the kernel. In Table 1, the 1.08μs (without ECHO processing) of PLATO are compared to the corresponding times in Section 5.2. *It should be noted that we are not merely comparing the time of a switch to the time of a similar function in software, but rather the time to take the data to the application and back*. In PLATO, the application *is* within the switch. This approach simplifies the network interface and speeds up the latency for data transfer to the application (run in hardware on PLATO), as shown in Table 1.

We understand that some published network interface measurements are for older technologies, however, Cornell's U-Net measurements correspond to quite recent network interfaces, and the 22.68 speedup cannot be affected substantially even with faster processors.

	Experiment	Time (μs)	PLATO Speedup
PLATO	ATM Cutthrough	1.08	1.00
TUC Measurements	UDP/IP	438.37	405.89
CORNELL's Experiments	Pure Device Driver	62.3	57.68
CORNELL's U-Net	Trap code	24.5	22.68

Table 1 PLATO SpeedUp vs. Network Interfaces (no application)

Speedup for measurements taken for an active node running the ECHO application (and in the case of ANTS for the similar IP forwarding application) on software vs. PLATO are shown in table 3. To compute the speedup, we used the overhead from the ECHO screening application running on PLATO, i.e. 1.53μs.

	Experiment	Time (μs)	PLATO Speedup
PLATO	ICMP ECHO Screening App. (ATM)	1.53	1.00
ANTS	Zero Payload Capsule, IP Forwarding (100Mbps Ethernet)	525	343.13
PLANet	Zero Payload packet, ping application (100 Mbps Ethernet)	100	65.35
ANN (SW Impl.)	IP Forwarding (ATM)	84.5	55.22
PAN	Kernel active x86, ICMP ECHO (100Mbps Ethernet)	70	45.75
	User Space Active x86 ICMP ECHO	133	86.92
	User Space Active Java ICMP ECHO	448	292.80

Table 2 PLATO SpeedUp vs. Software Approach on ECHO Application

5.5 Flexibility vs. Speed

All the above comparisons show how PLATO can improve performance over software approaches for

specific applications. Several arguments could be established regarding the technology of the available measurements in the software approach (no results for more recent technology are known to us). There are other arguments too, e.g. about Java flexibility vs. hardware flexibility. Java is certainly much more flexible than any kind of hardware, but the available resources on a recent generation FPGA allow for a large amount of area to be used for useful processing. Indeed, the entire 4x4 switch *plus* the ECHO application take less than 15% of the Virtex resources. Even with a PCI interface and the SDRAM controller, roughly 70% of the Virtex can still be used for payload processing. Through reconfiguration, additional applications can run on PLATO, including Wormhole IP over ATM. It seems apparent that the use of reconfigurable resources allows for a broad range of applications to be performed in real-time, and the lack of flexibility *vis a vis* a pure software approach is offset by the performance gains, whereas the reconfigurable processing is much more flexible than a VLSI approach.

6. Present Status

The Virtex-based PLATO platform has been designed at the Technical University of Crete (TUC). The 560 pin BGA package necessitated an 8-layer PCB design, which was fabricated by INTRACOM SA, a large electronics manufacturer in Greece. Assembly of the board, and the design of the UTOPIA daughterboard took place at the Institute of Computer Science (ICS)–FORTH. A PCI interface for PLATO has been designed at the Technological Educational Institute (TEI).

Substantial prototyping of the 4x4 ATM active switch core has been done with the PCI Pamette [32]. The prototype design has been successfully ported (in four person-days) to the actual PLATO hardware, using the PCI Pamette as I/O for PLATO. In addition, preliminary results for "ping" detection (i.e. processing of the payloads of ATM cells containing ICMP ECHO commands) have been fully simulated for the Virtex part and were found to run well (post place and route simulation) at 60 MHz. The "ping" application will form the basis for future DoS attack screening applications.

The ALTERA 20K400 (650-pin) version of the system has been designed at TUC and ICS, and its 10-layer printed circuit board is in fabrication by INTRACOM SA. The connectionless IP over ATM has been simulated at ICS for the ALTERA implementation.

Two more applications which are in advanced stages of development, are: (a) a dynamic cell routing scheme, in which routing is based on the payload of a cell, and, (b) a TCP/IP priority enforcement scheme based on the IP priority field. These applications are aimed at e-commerce speedup, as at present the routing of requests is done by servers which run the entire protocol stack, with all the performance limitations that were presented in section 5.

7. Conclusions and Future Work

PLATO, a new, reconfigurable platform was designed and implemented for experimentation with active networks. Initial results are very encouraging, both in terms of speed vs. traditional approaches and in terms of available FPGA resources for applications after the low-level protocol operations are performed. Over the next year we will continue to develop reusable code and develop more complex applications. Our goal is not only to achieve high performance vs. traditional approaches, but to have entirely new and unavailable to date capabilities as well, including DoS attack screening, connectionless wormhole IP over ATM , and network-based, real time, continuous speech recognition.

Acknowledgements

This work was supported by the Greek Secretariat for Research and Technology (GSRT) and the European Social Fund through the PENED 99 program under contract 99EΔ 408. We thank the Xilinx Corporation and the ALTERA Corporation for significant donations to our institutions. We are indebted to Dr. Laurent Moll and Dr. Mark Shand of Compaq Labs for the loan of a PCI Pamette, and for generous allocation of their time to transfer their PCI Pamette know-how to us. Lastly, we thank Mr. George Kalokairinos and Mr. Michalis Ligerakis for their invaluable assistance in the design of the printed circuit board.

Bibliography

[1] L. Moll, M. Shand, "*Systems Performance Measurement on PCI Pamette"*, In *Proceedings of FCCM `97*, pp. 125-133, April, 1997.

[2] T. McDermott, P. Ryan, Mshand, *et al.,* A Wireless LAN Demodulator in a PAMETTE: Design and Experiences, In *Proceedings of FCCM `97*, pp. 40-45, April, 1997.

[3] J. McHenry, P.Dowd, T. Carrozzi, *et al*., "An FPGA-Based Coprocessor for ATM Firewalls", Proceedings, FCCM `97, Napa, California, pp. 30-39, April, 1997.

[4] D. Wetherall, U. Legedza, and J. Guttag, "Introducing New Internet Services: Why and How", *IEEE Network Magazine*, July/August 1998.

[5] L. Lehman, S. Garland and D. Tennenhouse, "Active Reliable Multicast", In *IEEE INFOCOM'98*, San Francisco, CA, March 1998.

[6] B. Schwartz, A. Jackson, W. Strayer, W Zhou, D. Rockwell, and C. Partridge, "Smart Packets for Active Networks", BBN technologies, In *Proceedings of the Second IEEE Conference on Open Architectures and Network Programming (OPENARCH'99)*, March 1999.

[7] D. Alexander, M Shaw, S. Nettles, and J. Smith, "Active Bridging", Proceedings of the *ACM SIGCOMM'97 Conference,* Cannes, France, September 1997.

[8] M. Hicks, P. Kakkar, J. Moore, C. Gunter and S. Nettles, "PLAN: A Packet Language for Active Networks", In *Proceedings of the International Conference on Functional Programming (ICFP '98)*, 1998.

[9] S. Merugu, S. Bhattacharjee, E. Zegura and K. Calvert, "Bowman: A Node OS for Active Networks", In *Proceedings of IEEE Infocom 2000,* March 2000.

[10] S. Bhattacharjee, K. Calvert and E. Zegura, "Self-Organizing Wide-Area Network Caches", In *Proceedings of IEEE Infocom'98*, San Francisco, CA, March 1998.

[11] A. Campbell, H. De Meer, M. Kounavis, K. Miki, J. Vicente, and D. Villela, "A Survey of Programmable Networks", *ACM Computer Communications Review*, April 1999.

[12] A. Lazar, K. Lim, and F. Marconcini, "Realizing a Foundation for Programmability of ATM Networks with the Binding Architecture", IEEE Journal on Selected Areas in Communications (JSAC), Special Issue on Distributed Multimedia Systems, Vol. 14, No. 7, September 1996, pp. 1214-1247.

[13] S. Vinoski, "CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments", *IEEE Communications Magazin*e, Vol. 14, No. 2, Feb 1997.

[14] A. Campbell, M. Kounavis, D. Villela, J. Vicente, K. Miki, H. De Meer, and K. Kalaichelvan, "Spawning Networks", *IEEE Network Magazine* July/August *1999.*

[15] R. Keller, S. Choi, M. Dasen, D. Decasper, G. Fankhauser, B. Plattner, "An Active Router Architecture for Multicast Video Distribution", In *IEEE INFOCOM 2000*, March 26 - 30, 2000.

[16] S. Choi, D. Decasper, J. Dehart, R. Keller J. Lockwood, J. Turner and T. Wolf, "Design of a Flexible Open Platform for High Performance Active Networks", *Allerton Conference*, September 1999.

[17] D. Decasper, Z. Dittia, G. Parulkar, B. Plattner, "Router Plugins - A Software Architecture for Next Generation Routers", In *Proceedings of SIGCOMM'98,* September 1998.

[18] I.Hadzic, J.M.Smith, W.S. Marcus, "On-the-fly Programmable Hardware for Networks", In *Proceedings of Globecom*, 1998.

[19] W. Marcus, I. Hadzic, A. McAuley, J. Smith, "Protocol Boosters: Applying Programmability to Network Infrastructures", *IEEE Communications Magazine*, vol. 36, no. 10, pp. 79-83, October 1998

[20] P. Stogiannos, A. Dollas, V. Digalakis, "A Configurable Logic Based Architecture for Real-Time Continuous Speech Recognition using Hidden Markov Models", *Journal of VLSI Signal Processing*, Kluwer Academic Publishers, vol. 24/(2/3), pp.223-240, March, 2000.

[21] P. Newman, G. Minshall, T. Lyon: "IP Switching: ATM Under IP", *IEEE/ACM Transactions on Networking*, vol. 6, no. 2, April 1998, pp. 117-129.

[22] M. Katevenis, I. Mavroidis, I. Mavroidis, and G. Glykopoulos, "Wormhole IP over (Connectionless) ATM", Institute of Computer Science (ICS), FORTH. http://archvlsi.ics.forth.gr/wormholeIP

[23] P. Gupta, S. Lin, N. McKeown: "Routing Lookups in Hardware at Memory Access Speeds", *Proceedings of IEEE INFOCOM'98*, San Francisco, CA USA, April 1998.

[24] A. Dollas, K. Papademetriou, C. Mathioudakis, E. Markatos, M. Katevenis. "Experimental ATM Network Intrerface Performance Evaluation". *Technical Report FORTH-ICS/TR-244*, February 1999.

[25] T. von Eicken, A. Basu and V. Buch. "Low-Latency Communication Over ATM Networks Using Active Messages", *IEEE Micro*, Feb. 1995, pages 46-53.

[26] M. Welsh, A. Basu, and T. von Eicken. "ATM and Fast Ethernet Network Interfaces for User-Level Communication", In *Proc. of Third IEEE Symposium on High-Performance Computer Architecture* (HPCA-3), San Antonio, February 1997.

[27] J. Moore and S. Nettles. ``Towards Practical Programmable Packets." Technical Report MS-CIS-00-12, Department of Computer and Information Science, University of Pennsylvania, May 2000.

[28] D. Wetherall. Active Network vision and reality: Lessons from a capsule-based system. In *Proceedings of the 17th ACM Symposium on Operating System Principles (SOSP'99)*, Kiawah Island, SC, December 1999.

[29] E. Nygren, S. Garland, and M. Kaashoek. PAN: A high-Performance active network supporting multiple mobile code systems. In *Proceedings of the Second IEEE Conference on Open Architectures and Network Programming (OPENARCH'99)*, March 1999.

[30] M. Hicks, J. Moore, D. Alexander, C. Gunter, and S. Nettles. PLANet: An active internetwork. In *Proceedings of the 1999 IEEE Conference on Computer Communications (INFOCOM'99)*, January 1999.

[31] J. Turner, G. Parulkar, D. Dehart, S. Choi. T. Wolf, B. Platter, R. Keller. Design of a High Performance Active Router. In *http://www.arl.wustl.edu/arl/projects/ann/*

[32] A. Dollas, D. Pnevmatikatos, N. Aslanides*, et al*., Rapid Prototyping of a Reusable 4x4 Active ATM Switch Core with the PCI Pamette, In *Proceedings of the 2001 IEEE International Workshop on Rapid System Prototyping,* Monterey, CA , June, 2001(accepted, to appear).