A Generic Programming Model for Network Processors

PhD First Year Report

Kevin Lee BSc (Hons) MSc
Supervisors: Dr Geoff Coulson and Prof Gordon Blair

Distributed Multimedia Research Group,
Computing Department,
Lancaster University

January 2004

Table of Contents

Table of Figures

1 Introduction

1.1 Overview

In recent years the usage of the Internet has grown, both in terms of number of users and the requirements of those users. The Internet and the multitude of varying networks connected to it were once used largely for non-real time traffic with low performance requirements like email or web traffic. Increasingly becoming the norm is the proliferation of applications requiring heavy bandwidth or low response time like live video streaming (i.e. [Real, 03]) or online-gaming (i.e. [XBOX, 03]). As well as these increased performance requirements, a new generation of applications require the ability to interact with the underlying (inter-) network infrastructure to setup, control and maintain their specific requirements. Such applications include voice and multimedia services like real-time Conferencing [Thinkofit, 03] and applications that require specific security constraints of the network (i.e. remote database access, [Oracle, 03]).

The changing requirements of applications and services are forcing an evolution of the protocols and standards which govern the Internet, including new protocols such as IPv6 [RFC2460], real-time protocols such as RTP and RTCP [RFC1889], and QoS technologies like DiffServ [RFC3086] and IntServ [RFC2815]. However the deployment of these new standards is an expensive task, often involving the redesigning of network hardware and the potential of network downtime because of upgrades and replacement. Hence there needs to be an improvement in the mechanisms used to upgrade and manage the software in the network infrastructure of the Internet. Such an improvement would include support for security, dynamic updating of software, self management of network nodes and per-application specific requirements.

Currently the Internet's underlying architecture consists of a relatively small number of very large bandwidth and low latency fibre-optic backbones. These are connected together by preconfigured specialised ASIC-based hardware, which is designed primarily for speed. This "Core" Internet solves the problem of demanding applications by maximising its bandwidth and using efficient algorithms for routing, for example over the years the core(s) have changed from copper to fibre, from router to switch and from 100Mbs to 1Gb to 10Gb and beyond [Intel, 97]. The rest of the Internet follows a long way behind the core in terms of bandwidth and in terms of the actual hardware being used mainly for economic and support reasons. As such medium to large access networks and ISPs have had to be much more parsimonious with the network usage. However, users are increasingly utilising applications which require more specific and volatile network requirements. In order to support these varying network requirements the trend has been to use flexible general hardware like PCs or low-end rack-mount servers as routers in these networks; this has led to a bottleneck being created as these systems are not designed for high bandwidth usage.

Network Processors are an attempt by a number of large network hardware vendors including Intel, IBM and CISCO to fill the need for low-priced specialised network hardware suitable for edge-networks and corporate networks as well as for general use. Network Processors are simply a piece of equipment with a number of network

ports and a number of packet processing facilities which can be programmed with the aid of a toolkit. They normally consist of a general RISC processor, a number of small dedicated packet processors and some specialised hardware like buffer managers or hash-engines.

There is a downside to current Network Processor technology; that they are notoriously difficult to program to do specific (or general) tasks. This is because of their idiosyncratic and heterogeneous nature which also inhibits any translation of skills or software between brands. Indeed, it is acknowledged that Network Processors often consist of elegant hardware designs which are stifled by the stumbling block of its programmability, the very central benefit is was supposed to benefit [NetSpeed, 03]. This report presents a proposal which focuses on improving the programming facilities of network processors whilst unifying the approach of programming different brands of Network Processors in order to increase the possibilities of design portability between devices.

Although more flexible in their usage than traditional ASIC hardware, Network Processors with the current generation of software are difficult to transform to perform a different set of tasks once it has been deployed (discussed in [InfoWorld, 02]). As such the proposed research also has the additional aim of introducing support for dynamic reconfiguration of network processor software in order to increase the flexibility of network processors which can be seen as vital to the increasing acceptance of network processors.

1.2 Standardization

There are a number of current efforts to standardise the development and programming of Network Processor in order to ease their use. One of the more promising standardisation efforts is being pushed by the Network Processor Forum. The Network Processor Forum is a organisation of manufacturers, research organisations, vendors and standards agencies who are attempting to solve some of the problems of current Network Processor technologies. It also identifies itself as being an organisation to "facilitate and accelerate the development of next-generation networking and telecommunications products based on network processing technologies" [NPForum, 03].

The major problems identified by the Network Processor Forum are the increased costs to hardware and software vendors due to the lack of standardisation and the lack of inter-operability between products. The Network Processor Forum attempts to solve these problems through increased co-operation, research sharing and standardisation between the leading "Network Processor-interested" corporations. The solutions are centred around reducing costs through standardisation. The forum defines both hardware and software standards and specifications; however there has been so far little take-up of the hardware designs.

The more successful aspects of the Network Processor Forum involve the defining of layered programming and service abstractions to which software writers and hardware vendors can design and implement their products to.

The forum has a number of actual successes including a complete IPv4 unicast forwarding service specification based on a generic API for Network Processors. The forum also has successful specifications on how to design compatible benchmarking tests for different Network Processor platforms. The forum is also attempting, with the help of its major members of IBM, Intel, Alcatel, Ericsson and IDT to standardise the API's through which Network Processor can be designed, something akin to the IBM-compatible de facto standard.

1.3 Reference Model

For the purposes of this report, each reviewed technology will be classified according to the reference architecture for programmable networking in figure 1 (appeared in [Coulson, 02]).

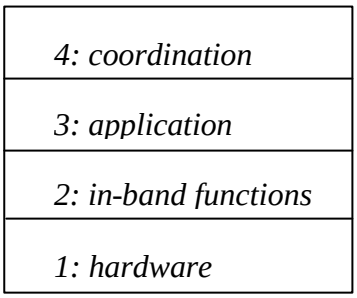


Figure 1.1 reference architecture for programmable networking.

This architecture contains four levels of abstraction called Strata which should not be confused with layers in a protocol stack. Stratum 1 in this architecture represents operating system functionality like threads, memory allocation and network access. These services are the minimal operating system functionality required to support the higher-level (higher-stratum) programmability. The Services exposed to higher levels by this stratum would typically present an abstract view of the underlying heterogeneity. For example the two different platforms of a standard PC-based router and an Intel IXP1200 [Intel, 00] could present the same abstract interface to higher stratum, whilst also exposing additional platform specific functionality.

The second stratum called "in-band functions" encapsulates all the fast-path functionally like packet filters, classifiers, DiffServ schedulers, traffic shapers and checksum validators. These functions will be performed on most packets almost routinely and hence are inherently low-level and fine-grained. Because of this, great care most be taken when developing or supporting software at this level, to ensure it performs well. The third stratum contains coarser grained applications which are less performance critical than the stratum 2 applications and act on pre-selected packet flows, for example per-flow media filters. The uppermost stratum is that describing the coordination stratum, which contains out-of-band signalling including configuration and reconfiguration. Examples of technologies in this stratum include RSVP and any protocols that coordinate resource allocation across sets of routers in a dynamic virtual network.

1.4 Structure of the Report

The underlying topic of this report is the lack of progress in current network processor software to support multiple programmable architectures and hence present a unified way of developing software for Network Processors. This report will also attempt to show that supporting dynamic reconfiguration of network processors is a important.

This document presents the main points of the research that the author has undertaken in the first year of his PhD, and finishes with the proposal for the remainder of the PhD. Chapter 2 introduces a survey of current Network Processor technologies, presenting the various commercial products and analysing their differences and similarities. Chapter 3 presents a survey of available Network Processor software; it first discusses the various commercial software packages and then analyses some academic research projects. Chapter 4 introduces the principles of component-based programming and then presents system-level component software, which I argue can be used as a basis for designing software for network processors. Chapter 5 presents a proposal for the remainder of the PhD, including a statement of aims and methodology, an evaluation framework together with a programme of work. Finally, Chapter 6 presents some conclusions from this report and sums up the proposed research.

2 Network Processors

2.1 Introduction

In this chapter we give the reader an introduction to the current state of the art in the field of network processors. We focus most on two of the most investigated and popular network processors, the Intel IXP1200 and the IBM PowerNP, and we also investigate a number of less well known network processors. We analyse their architectures, focus on their individual intricacies and evaluate their positive and negative points. The final section of this chapter will draw some conclusions about the various network processors and try to sum up the general trends in the field.

2.2 The Intel IXP1200

The Intel IXP1200 [Intel, 00] is a highly integrated, high performance, asynchronous data processor which is targeted at delivering "high-performance parallel processing power and flexibility to networking products". The IXP1200 is part of Intel's Internet Exchange Architecture [Intel, 03] product series which combines "unlimited programmability" and "robust packet handling performance" to support the rapid development and deployment of intelligent network services, while helping to extend product lifecycles for lower total cost of ownership.

The IXP1200 processor itself consists of a single embedded RISC processor (an Intel Strong-Arm), and 6 (depending on the version) micro-engines. Any tasks which are run on the IXP1200 are normally split into two parts, the control plane and the data plane, or out-of-band and in-band. The out-of-band control process and any exceptional traffic runs on the slower but more general Strong-Arm, whereas the in-band non-exceptional traffic runs on the micro-engines, which are packet processors.

The IXP1200 was specifically designed to be a network processor, and as such usually comes in the form of a single PCB with the central IXP processor and a number of other components enable a complete network processing platform. This is illustrated in figure 2.1 which shows the relationship between the components of the IXP platform. The diagram shows the various entities which work with the IXP1200, including Network Interfaces to send and receive packets, the various types of memory and the two buses used to allow the components to communicate.

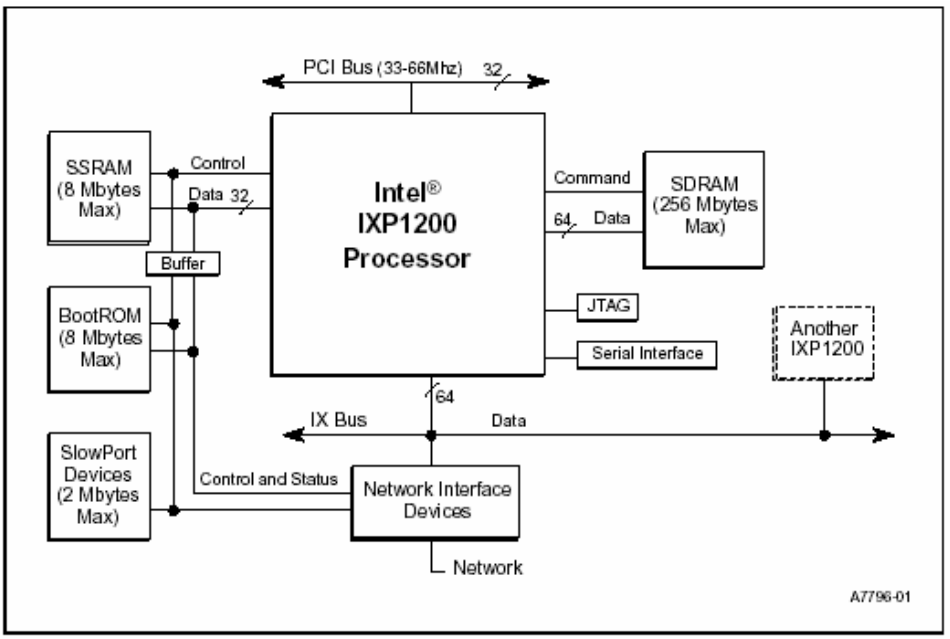


Figure 2.1 IXP1200 System block diagram (from [Intel, 00])

The IXP platform also has a number of different types of memory, which are useful when performing different tasks with the platform. The different types of memory include scratch pad, which is a small amount of on-die memory used for program data, SRAM which is useful for storing large data structures for fast access like routing tables, and SDRAM which would ordinarily be used for buffering packets.

The IXP1200 also has a number of important features, these are:

- A 32-bit PCI interface allowing interfacing with industry standard PCI devices.
- An asynchronous serial interface for use as a debugger console over RS-232.
- A Boot ROM which allows booting of the platform with a number of popular operating systems including VXWorks and Linux.
- Hardware coprocessors for specific tasks like CRC checks.
- Optimized instruction set for efficient creation and use of functions related to networking.

The IXP1200 network processor development package comes in the form of a PCI development board and a large software and manual package. The default development platform is the Intel ace architecture which runs on a bootstrapped Linux system on the StrongARM processor. The Intel ACE architecture is discussed in section 3.2. There is also a number of commercial toolkits available for the Intel IXP1200 including the TEJA development platform discussed in section 3.6.

The Intel ixp1200 is currently the most researched networked processor available, and as such there are a number of semi-mature development platforms, helper utilities and specific code available.

2.3 The IBM PowerNP

The IBM PowerNP Network Processor [IBM, 03b] is an extremely powerful and scalable Network Processor architecture. It consists of a diverse and extensive range of embedded processors and coprocessors as well as a number of smaller specialised functional units.

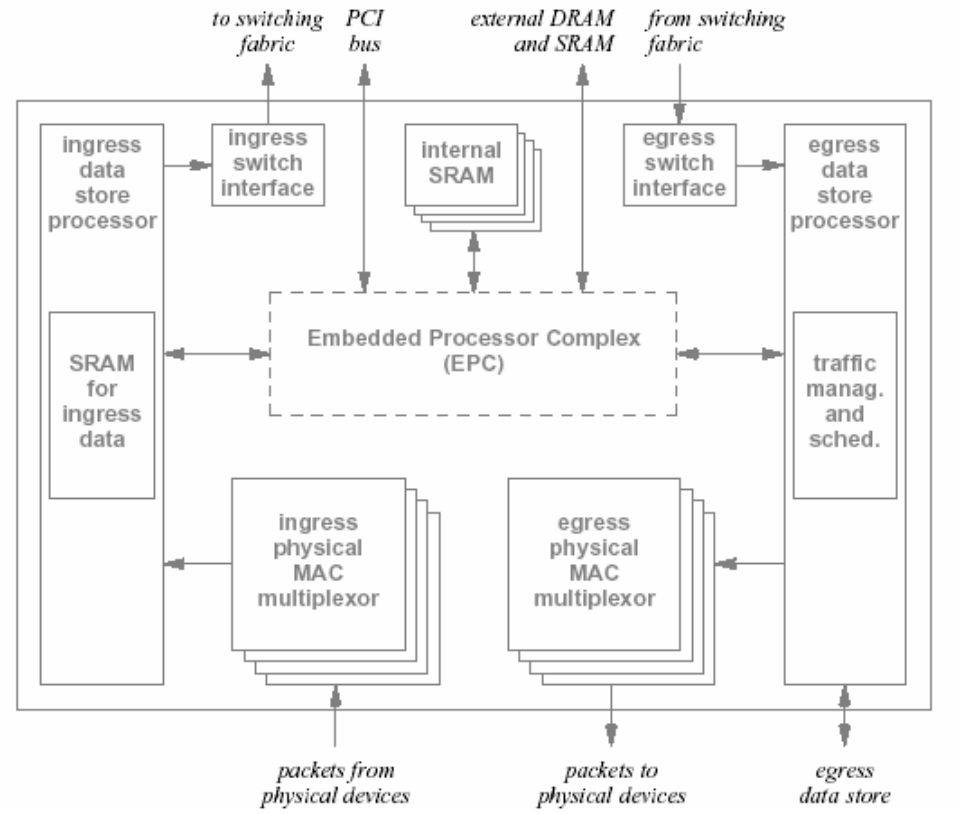


Figure 2.2 Internal Architecture of IBM PowerNP (from [Comer, 03])

Figure 2.2 (from Comer 15.12) illustrates the general architecture of the IBM PowerNP. It shows that the architecture is based around a central point called the Embedded Processor Complex (EPC) which is surrounded by hardware components to enable packet processing. The Embedded Processor Complex contains a number of processors and coprocessors. This includes sixteen Multi-threaded Dyadic protocol Processing Units (DPPU) called "Pico-engines". According to [IBM, 03a] the term dyadic "refers to the two core language processors that each DPPU contains". Each DPPU also controls four threads which means each DPPU unit can be doing 4 processors for each core language processors, hence 8 tasks per DPPU and 64 tasks in total.

Also on the Embedded Processor Complex there are 10 shared co-processors for use by the core language processors. There is 1 co-processor data bus for transferring data to and from the co-processors and 1 co-processor command to allow the core language processors to send instructions to the co-processors. The Embedded Processor Complex also has 32 Kbytes of instruction memory for the eight DPPUS.

As well as the DPPU units there are a number of special purpose co-processors, these are:

1. Data Store co-processor, which provides a frame buffer Direct Memory Access
2. Policy co-processor, which performs programmable traffic management
3. Enqueue co-processor, which passes packets to switch or output queue Counter, which updates logging counters when processing packets
4. Interface co-processor, which provides a programmable interface to DPPU registers and memory
5. Semaphore co-processor, which manages threads.
6. String Copy co-processor, which transfers large volumes of data at high speed
7. Checksum co-processor, aids in checking checksum bit in headers

2.4 The EZchip NP-1

The NP-1 [EZchip, 03] is a network processor designed by EZchip Corporation. It uses special purpose heterogeneous processors arranged in a pipeline. Each processor is designed to handle a specific task. Figure 2.3 illustrates the architecture of the EZchip NP-1, the four different processor types are:

- TOPparse, which handles header field extraction and packet classification
- TOPsearch, which handles routing table lookup
- TOPresolve, which handles buffer management and packet forwarding
- TOPmodify, which handles any changes which need to be performed on the packet

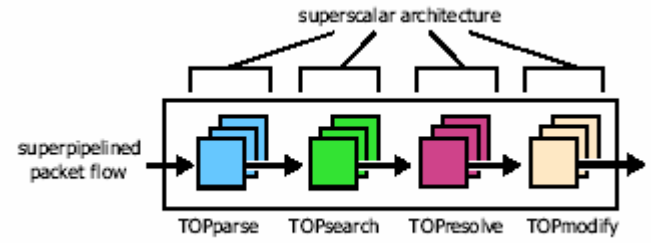


Figure 2.3 Architecture of EZchip NP-1 (from [EZchip, 03]])

The EZ-chip NP-1 differs from the more mainstream IBM and Intel Network Processors because of its unique all-pipeline design. This design attempts to provide all the necessary facilities of a network processor in a highly restricted design, whilst still enabling flexible software design.

2.5 The Motorola C-Port

The Motorola C-Port (C-5, C-5e and C3) [Motorola, 03] is a more flexible design than most other Network Processors as its general architecture consists of a single chip with a number of functional units which can be arranged in a pipelined or parallel manner depending on the requirements of the task. This is achieved by using a switching fabric illustrated in figure 2.4 consisting of multiple onboard buses and clusters of functional units.

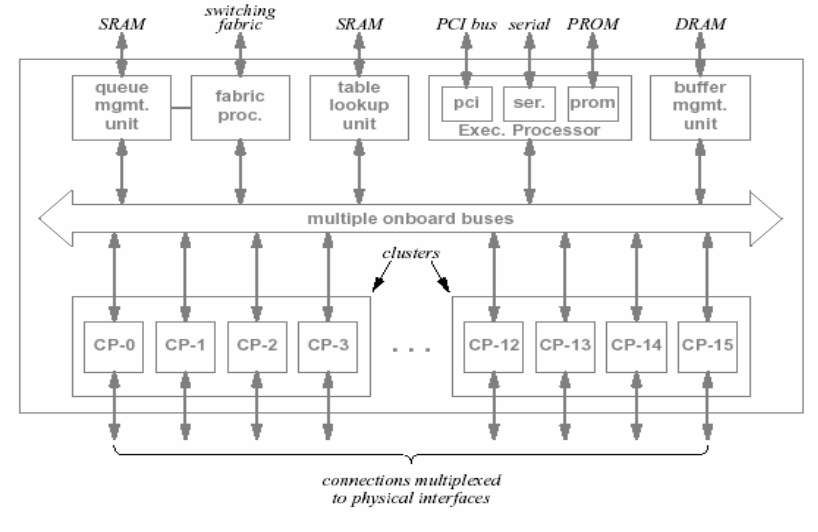


Figure 2.4 Internal architecture of Motorola C-Port NP (from [Comer, 03])

The Motorola C-Port design is very similar to that of the IBM and Intel Network Processors which have a number of different functional units connected together by a bus. It differs from those designs by utilising an efficient switching fabric reminiscent of more expensive ASIC hardware in order to connect the functional units to the system bus. However it is clear that utilising such a design where each functional unit is independent and connected to other functional units by complex independent switching fabrics could be difficult to develop software for.

2.6 The Agere PayLoadPlus

The PayLoadPlus [Agere, 03] Network Processor manufactured by Agere Systems offers a number of unique features including multiple-processor architecture, emphasis on ingress processing and the use of a programmable classifier.

Figure 2.5 shows that the PayLoadPlus consists of three separate chips which work together to efficiently process packets, the diagram also shows the flow of packets through the system. The Fast Pattern Processor is a classifier for incoming packets and the routing switch Processor uses the information from the classifier (FPP) to determine which direction the packet takes through the switching fabric and ultimately which network to be sent to. The Agere System Interface contains a number of coprocessor function units which supply important functionality to improve performance.

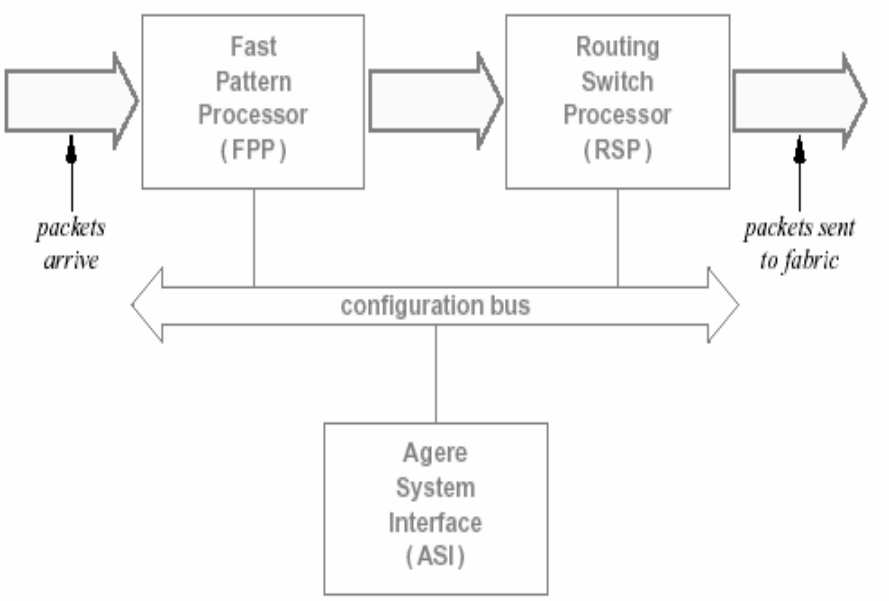


Figure 2.5 System architecture of the Agere PayLoadPlus (from [Comer, 03])

On one the main advantages that the PayLoadPlus has over the competition is its ease of programmability. The PayLoadPlus contains a highly advantaged architecture consisting of pipelines, threading and contexts similar to other Network Processors. However unlike other Network Processors the PayloadPlus succeeds in hiding a lot of the complexity of this parallelism from the application programmer. It does this by offering a high-level classification language called "Functional Programming Language" and a scripting language called "Agere Scripting Language".

2.7 Cisco Parallel eXpress Forwarding (PXF)

The Cisco Parrallel eXpress Forwarding (PXF) [CISCO, 03] network processor [CISCO, 03] is different than the other Network Processors reviewed in this report because it is built using ASIC hardware. Using a ASIC design involves utilising a array of components and fixing the data paths through the device at manufacture time. So using ASIC hardware has the advantage of increased performance than programmable platforms, but has the distinct disadvantage of lack of post-development flexibility and higher initial costs.

Figure 2.6 illustrates the strategy of Cisco Parallel eXpress Forwarding Network Processors to use parallel pipelines of homogeneous processors. A single chip contains 32 embedded processors which can be arranged in 4 parallel pipelines, which can then effectively perform operations on four different types of traffic.

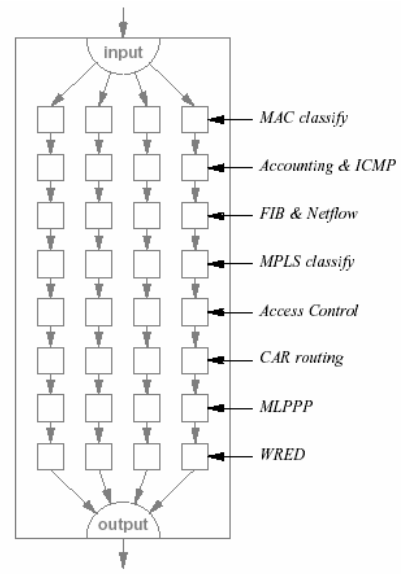


Figure 2.6 Cisco PXF Network Processor architecture (from [Comer, 03])

The CISCO PXF Network Processor is extremely similar in design to the NP-1 by EZchip in that it is a highly restrictive pipelined design of specific functional units. There is however a crucial design difference between the two products, the EZchip Network Processor restricts itself to a single pipeline; however the CISCO Network Processor contains a number of different pipelines. The replication of pipelines in the CISCO PXF introduces a number of new possibilities, but also adds some software design problems. For example, the EZchip design can assume that the pipeline will either process or drop each packet it receives; the CISCO design however must also decide which pipeline processes each packet.

2.8 Analysis

As can be seen from the reviews in this chapter, the world of Network Processors is wide and varied. The Intel IXP series and IBM NP series of Network Processors are the most advanced and most supported Network Processor platforms available. They are also the most flexible when compared to the other Network Processor designs, which are very restrictive in how software is created for them. The other more restrictive platforms like the CISCO and Motorola solutions do have their advantages though, because they are built using specific programmable hardware rather than the general processors and packet processors used by the IBM and Intel Network Processors, they are more straightforward to develop software for and also more efficient.

Because the network processor platforms themselves are so different and because as we shall see in the next chapter the software development kits are so different, there is very little cross-over in software strategies, reuse or standardization. However, the Network Processor designs featured in this chapter so have commonalities in their design and functionality; figure 2.7 illustrates these commonalities. For example they all have some form of specialised packet-forwarding engine, all have hardware buffer support and a variety of hardware co-processors. There are some major differences of

architecture, for example the choice of whether or not to have a general-purpose processor and the choice of co-processors. Each Network Processor has a different type of packet forwarding engine, microengines, picoengines, channel processors and the more exotic forwarding engines of the agere and cisco Network Processors, which effectively do the same task of programmably forwarding packets at line rates. The IBM and Intel Network Processors both have General-purpose processors, although the others also have mechanisms to allow general processing, i.e. utilising a host computers CPU. The different Network Processors use different terminology to describe the process of buffering, queuing and forwarding packets; all the Network Processors contain specialised hardware to perform these tasks. One area of wide variance between the different Network Processors is in there chosen use of hardware co-processors, with a number of different types including table lookup units, hash table units, tree searching and frame alteration.

The main point is that they are all capable of performing exactly the same task in an efficient manner, differing in only the task implementation not in what is achieved. Furthermore, at the packet processing level, all the reviewed Network Processors have no features which inherently prevent dynamic reconfiguration from occurring, and as we will see in the next chapter it is inadequately designed software which prevents dynamic reconfiguration.

	Packet-Forwarding engines	General-purpose processor	Buffer management Support	Co-processors
Intel IXP Range	microengines	StrongARM+x86	Multiple buffers	Hardware hash unit Various management units
IBM PowerNP	picoengines	PowerPC	Traffic Management Scheduling Ingress and Egress Switching fabric	Tree searches Frame forwarding Frame filtering Frame alteration
EZchip NP-1	Processing engines	No	Queuing fabric	Hardwired optimized algorithms on single chip
Motorola C-Port	Channel Processors	No	Buffer Management Unit Queue Management Unit	Table Lookup unit

Agere PayLoadPlus	Fast Pattern Processor Routing Switching Processor	No	Buffer and queue engines	CRC engine ALU Pattern engine
Cisco Parallel Plus	Four pipelines of homogeneous processors	No	Specialised embedded processors	Specialised embedded processors

Figure 2.7 Tabulation of Network Processor commonalities

By observing the commonalities between Network Processors and also remembering to respect the fundamental differences we can also draw a theoretical model of a Network Processor. Figure 2.8 from [Comer, 03] illustrates this theoretical model with respect to packet flow through a Network Processor. The main idea is that the more complex the packet processing requirements the more general processing engines are used. For example the majority of traffic is processed at the lower levels of the processor hierarchy, the more complex or exceptional the packets become the more that they have to be processed by processors in higher levels of the hierarchy.

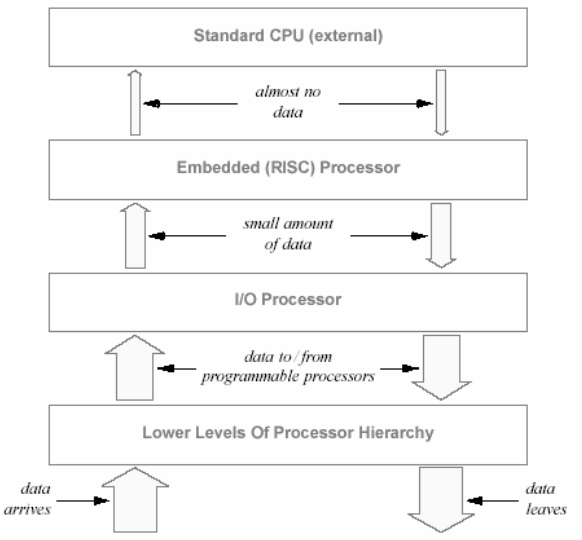


Figure 2.8 Illustration of packet flow through a Network Processor

The reviews of the different network processors in this chapter has illustrated that one of the main tradeoffs and design considerations with Network Processors is between providing a highly optimized hardware assisted design whist attempting not to over-complicate the programmability of the platform. As we will see in the next chapter, this has led to differing techniques for software development kits depending on the target platform.

3 Network Processor Software

3.1 Introduction

This chapter details the commercial software available for some of the network processors discussed in the previous chapter. It tries to give a general picture of the abilities and the failures of this software. This chapter also presents some research (from section 3.5) aimed at enhancing the programmability and configurability of network processors. The Chapter ends with an overall analysis of Network processor software and tries to sum up the general trends in the field and future directions.

3.2 Intel ACE/MicroACE

As well as the usual development environment, debuggers, emulation and deployment software which you would expect of any programmable network device, the Intel IXP1200 development kit also provides a programming model called ACE. ACE is a concept defined by Intel, and refers to a unit of programming abstraction in the IXP1200 called an Active Computing Element.

An ACE can be characterised as: (from [Comer, 03])

- Fundamental software building block
- Used to construct packet processing systems
- Runs on StrongARM, microengine or host
- Handles control plane and fact or slow path processing
- Coordinates and synchronises with other ACEs
- Can have multiple outputs
- Can serve as part of a pipeline

Essentially the ACE has been envisioned as the core building block for developing software solutions on the IXP1200. It is a specification for a module which can communicate with other modules running in various locations on the IXP1200. It can also handle different types of traffic, both Control and data path traffic can be processed within an ACE module.

As is fitting of a one solution fits all design, the ACE actually comes in different varieties. If for example a developer was designing an IP software package based on the IXP1200 and its facilities, the ACE system design would look something like figure 3.1. The diagram introduces the concept of a MicroACE, which are the three lower ACEs in the diagram. A MicroACE has exactly the same specification as a standard ACE except that it contains a microblock, which is a block of specialised code which runs on the microengines of the IXP1200. A MicroACE is in contrast to standard ACEs (the upper 4 ACEs in the diagram) which reside entirely on the StrongARM processor.

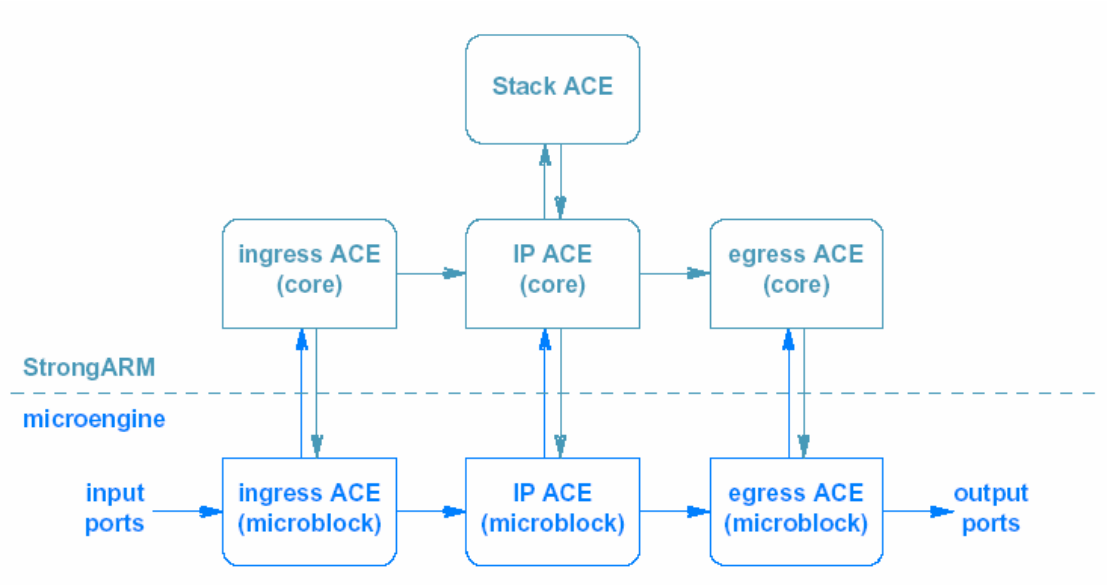


Figure 3.1 an Example ACE System (from [Comer, 03])

The different ACEs communicate by utilising the various mechanisms available on the board. For example the MicroACE modules communicate with the StrongARM ACEs by utilising the internal bus called the IX Bus (present in figure 3.1). The IX Bus also allows the MicroACE and ACE modules to access the various control processors and memory types.

The diagram shows that the main bulk of the traffic and the processing occur in the microengine layer, which includes ingress and egress traffic management as well as classification and in-band packet processing. The MicroACE modules handle only straightforward traffic which they understand; usually any exception cases are taken out of the microengines domain and forwarded to the ACEs running on the StrongARM processor.

The development of the ACE framework for use with the IXP1200 Network Processor makes it considerably easier to develop complex software solutions for the platform. Previously all development had to occur by hand-coding the individual blocks of code and then statically binding them together in a monolithic block and assigning sections of code to specific processors and functional units.

There are however shortcomings in the design and functionality of the ACE programming model. Although the general architecture appears component-based, it has only some of the advantages of a component-based programming model (see chapter 4). These are limited mainly to easing the initial learning curve of the IXP1200 and also making easier to develop straightforward software. When more complex or efficient software is required, the systems programmer must still grapple with the details of the IXP1200 hardware. The ACE framework also does not provide support for dynamic reconfiguration at run-time, which means that when the requirements change for a particular device and its software the system would have to be stopped and reloaded with the new program and restarted.

3.3 IBM PowerNP Developers Toolkit

In order to encourage acceptance of the PowerNP platform IBM has made available an extensive range of programming support. These are illustrated in figure 3.2. The diagram illustrates the usual set of tools including compilers and assemblers as well as simulation and debugging packages which aid in easing the difficult task of programming a Network Processor.

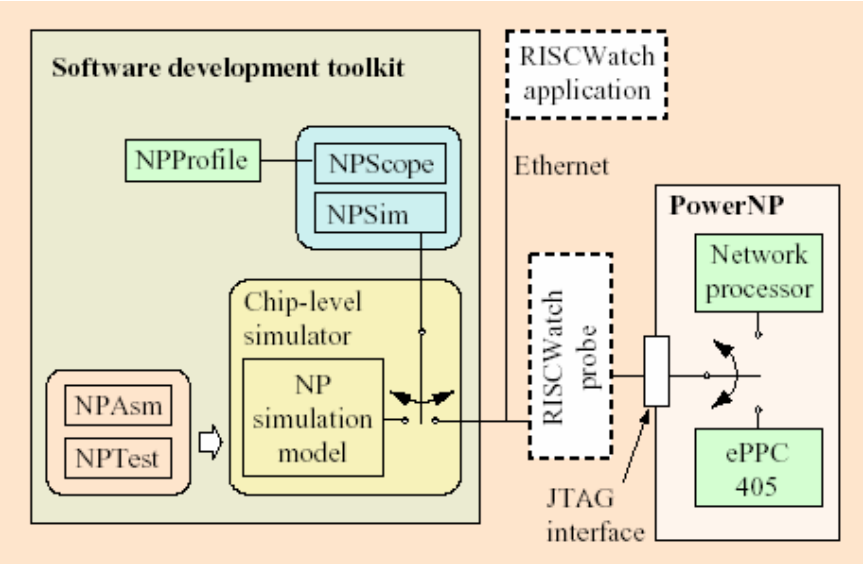


Figure 3.2 IBM PowerNP software development toolkit (from [IBM, 03b])

IBM has attempted to make the PowerNP more attractive than other Network Processors by providing an extensive implementation set with its advanced software package. The implementation set includes IPv4 forwarding, Multicast, Diffserv and MPLS; there is also a comprehensive API for configuration and control.

IBM claims in [IBM, 02] that the software in the advanced software package is portable to other platforms with ease because of its design in a layered manner. Similarly to the Intel IXP1200 ACE framework, the IBM SDK contains all the necessary development tools, including a compiler/assembler/debugger and simulator.

IBM's approach of software offerings is strikingly different than Intel's approach; whilst Intel's approach is to provide sets of rules and frameworks to simplify the systems programmers' task, the IBM approach is to structure the development lifecycle itself by providing software building blocks with which software can be assembled. Thus, a Intel software developer will actually program the Network Processor, whereas an IBM software developer will mostly assemble and configure IBM software building blocks. The IBM approach could be argued to be less flexible than the Intel approach due to the programmer not necessarily knowing the details of the hardware itself, however the time saved using pre-packaged software can be used to optimize any solution.

3.4 NetBind

NetBind is a tool for dynamically binding components into a data path in network processor-based routers. The architects of NetBind describe it as "a high performance, flexible and scalable binding tool for dynamically constructing data paths in network processor-based routers" [Campbell, 01]. NetBind uses a unique technique to create data paths in Network Processors. Instead of requiring the programmer to include some "magic" code in order to allow the binding to occur, NetBind simply modifies the machine language code of pre-compiled components when binding is required.

NetBind attempts to balance the flexibility of network programmability against the need to process and forward packets at line speeds. Data paths constructed using NetBind seamlessly share the resources of the same network processor. The system focuses on the fact that the higher line rate supported by a network processor-based router means that typically there is a smaller set of instructions that can be executed in the critical data path. Therefore, in order to make the constructing of data paths as efficient as possible, NetBind keeps the overhead of binding down to a minimum.

Some of the more interesting research by the NetBind team is documented in [Kounavis, 03b] which focuses on analysing the performance of different packet classification algorithms on the IXP1200.

This process of dynamically generating machine code reduces the number of binding operations required for constructing data paths that are normally required for dynamic binding. In fact for fast path data composition the number of binding operations is so reduced that it is comparable to packet forwarding times. However, the work is heavily biased towards the Intel IXP1200 Network Processor, and does not take into account other Network Processors. This is mainly due to that fact that NetBind is extremely low level software, mostly dealing with assembly instruction manipulation. So to port NetBind to another platform would require a complete re-write of almost all the IXP1200 codebase.

The current NetBind implementation focuses on the Intel IXP1200 network processor discussed in section 2.2. It consists of a set of libraries that can interface with the IXA1200 micro engines, modify the IXA1200 instructions, create processing pipelines and can perform higher level operations.

The NetBind system address's the gap between using synthesised kernel code for constructing modular paths, and general-purpose processor architectures. The systems binding technique is optimized for network processor-based architectures and minimises the binding overhead in the critical path and thus allowing network processors to forward packets at line rates.

NetBind can be classified as being in strata 2 of the reference model discussed in section 1.4. This means that in any software architecture, NetBind would provide its services to strata 3 and 4 of the model which are the application and co-ordination software.

3.5 VERA

VERA is an Extensible Router Architecture which attempts to address two issues, that of the increasing number of services a router must support and the increasing diversity of router hardware which is available, by developing a system which is hardware independent and allows extensible services.

The architecture aims to be extensible by allowing "new functionality to be easily added to the router" [Karlin 01]. It also aims to be a fully compliant router by supporting RFC1812 [Baker, 95] which defines a set of criteria for IP version 4 routers to support. As with other router architectures, VERA also aspires to be highly efficient on the particular implementation hardware.

The VERA system itself consists of "a router abstraction, a hardware abstraction, and a distributed router operating system" [Karlin 01]. The router abstraction provides an interface which the actual routing functions can be performed. The hardware abstraction provides just enough hardware details for the router functions to be performed without making the router architecture hardware dependent. The distributed router operating system sits between the hardware abstraction and the router abstraction providing a mapping between the two and also managing communication between the abstractions.

Currently VERA has been implemented on a single processor architecture which employs a number of Network Interface Cards (NICs), and has been used to implement a number of different types of routers. It has also been partially implemented on the IXP1200 where they claim to guarantee line speed for simple packets whilst still having capacity to classify and process exceptional packets on the Pentium processor [Spalink, 01].

The Distributed Operating System contains a number of interesting features. The network processor based System's organisation is based around a processor hierarchy, where different levels of packet processing time is executed on different levels of the processor hierarchy. The initial classification occurs on the processor controlling the network interface, if the processor requires further or more complex processing then a higher level processor will be used. The master processor controls the system and deals with any resource reservation or processing overflow. This type of architecture at the core of the operating system should make the task of deploying the system on different architectures.

The Thread assignment and scheduling takes place on a per flow basis called a "path" by the authors and initialised by the "createPath" function. This clearly supplies core support for Quality of Service based threading mechanisms. Another interesting feature is the use of a packet header to describe the details of each packet and to reduce the number of expensive copy operations.

The VERA platform can be classified as being within stratums 2/3 of the reference model, this means that the bulk of its functionality is application-level which manages the higher-level mechanisms whilst controlling the in-band functionality.

3.6 S3 and Teja NP

The commercial Network Processor software field is being lead at this point in time almost single-handedly by the alliance between Silicon & Software Systems (S3) and Teja Technologies.

Silicon & Software Systems [S3, 03] is an established Electronics Design Company which was founded in Dublin, Ireland in 1986 and specializes in System Level Design and Integration. One of S3's target development areas is Network Processors. S3s mission statement is to "Reduce the complexity, risk and expense that equipment manufactures face when building complex network elements using network processing technology".

Teja Technologies [Teja, 03] is a software development company based in San Jose California; it is backed by a number of large companies including Intel. It has the aim of "providing a complete software development platform for network processors".

Together these companies are targeting their development at the two most successful network processors, the Intel IXP series and the IBM NP series. They have a number of emerging successful Network Processor technologies for development, analysis, deployment and management. The main software package being developed is the Teja NP platform. The Teja NP package promise to "greatly accelerate product development, providing network equipment vendors with the time-to-market advantage, high performance, reduced development costs, and greater flexibility associated with using network processors".

The Teja NP package claims to be the first complete platform for Network Processors. The platform claims to achieve the following results:

- Provides an environment for creating network processor applications in less than half the time
- Efficiently manages the resources of the multiple processors in a network processor
- Efficiently manages the resources of the multiple processors in a network processor
- Optimizes the performance of parallel architectures
- Provides a rich set of APIs for resources and service management to meet the specific requirements of embedded network applications
- Gives you compatibility with legacy standards and operating systems, such as VXWorks and emerging (embedded Linux) RTOSs.

The Teja NP platform appears to make the development and management of Network Processor software a more straightforward and structured task.

3.7 Analysis

Both the Intel ACE/MicroACE framework and the IBM PowerNP SDK provide powerful tools for developing solutions for the respective platforms. In addition they also provide advanced simulation tools to develop and debug solutions without the overhead of testing on the hardware itself. The main aims of these solutions is to provide the tools and development environments to aid the systems programmer to develop for the platforms, not actually provide innovative ways of developing software.

NetBind provides an efficient mechanism to dynamically bind microcode within the microengines of the IXP1200 Network Processor, however it does not provide any higher level mechanisms to support this and hence is extremely difficult to use. So although NetBind gives some support to enable dynamic reconfiguration, it

So it is clear that the advanced development and simulation tools are available and mature, and there are also frameworks available to aid in structuring software solutions. It is also clear that the research community are tackling some of the more interesting issues of dynamic binding and router abstractions. However there is a serious lack of development platforms with the advanced features like dynamic bindings (see section 4.2.2) supported by advanced tools and frameworks.

Also, the current generation of 3rd-party Network processor software is focusing almost entirely on the Intel and IBM hardware, mainly because of the research campaigns started by those companies, but also because these are the simplest platforms to develop for because of the inclusion of a central processor. There are equally powerful Network Processors with different types of design, as illustrated in chapter 2, however there is little commercial or academic software being developed for these other Network Processors.

The work of the Network Processor Forum is going some way towards increasing inter-operability between platforms and thus aiding in increasing the programmability of the field as a whole. Although as with most standardisation work, it will take time to filter through into vendors products and commercial and academic software packages.

So, the main issues identified in this chapter are:

- Lack of generic models and frameworks across Network Processors.
- Lack of high performance Network Processor software which is also flexible.
- Lack of dynamic reconfiguration support in Network Processor Software.
- Lack of high-level programming language and programming model support for low-level facilities.

4 Systems-level Component Software

4.1 Introduction

This chapter concentrates on innovative research which utilises component models to achieve their aims on a number of domains in the active networking field. Firstly, section 4.2 will introduce the general field of component software, and discuss component models component frameworks and computational reflection. The chapter will then presents some relevant research in other related fields including component-based kernels and general component software. The commonality of the research in this chapter is that they all use components to try to increase the configurability or flexibility or ease of programmability in a field where timing is critical, i.e. embedded kernels or programmable networking software. At the end of the chapter there is a final analysis of systems-level component software, summing up trends of the research in this chapter and finally drawing some general conclusions.

4.2 Component software

This chapter gives a brief overview of the main concepts of component models. Firstly component-based programming is generally described, and then details of component models including interfaces, components and binding mechanisms are detailed. The important strategy of using Component frameworks is discussed before finally focusing on computational reflection.

4.2.1 Component-Based Programming

Component Based programming is a type of programming whereby implemented systems are built up of a number of discrete modules called components. These individual components each perform a discrete task, or each provide a specific function that the system as a whole needs. A component is a small part of a larger system performing a unified task.

The use of components in the programming of a system has a number of advantages over other methods of programming. These include the ability to upgrade components, interchange different versions or replace components with new implementations.

Examples of types of components include graphical interface components like menus and buttons, modular operating system components like memory managers or device drivers or distributed object software like CORBA.

4.2.2 Binding Mechanisms

"Binding" describes the way in which two components are connected together to perform an operation. There are two types of binding in component-based systems, a static binding and a dynamic binding.

A static binding is a type of binding which is fixed when the system is compiled, assembled or deployed. Essentially the important thing about static bindings is that once a component has been statically binding there is not normally the possibility of dynamically rebinding it at runtime without restarting the system.

This is the crucial difference between static and dynamic bindings, that static bindings cannot be reconfigured at runtime whereas a dynamic binding can be altered at runtime. A component model which enables dynamic bindings has mechanisms which allow bindings to be created, deleted and recreated dynamically at runtime.

4.2.3 Component Frameworks

A Component Framework is a "collection of rules wand interfaces (contracts) that govern the interaction of a set of components plugged into them" [Szyperski, 98]. A specific Component Framework is targeted at a specific domain and embodies "rules and interfaces" that make sense for that domain. For example a component framework for a router would embody "rules and interfaces" for routing, buffering and processing a specific type of packet, i.e. IP packets.

Component Frameworks are 'life support environments' for 'plug-in' components and help in configuring and dynamically reconfiguring areas of the system. They define the software architecture, a set of components and their interaction mechanisms. A dynamical reconfigurable component framework allows the reconfiguration of its set of components at runtime without having to shut down the system and risking the integrity of the network as a whole.

Component Frameworks are used as a tool to structure software in a wide range of fields. They can be used to develop Application software including web browsers and Integrated Development Environments (IDE') like those built using the XPCOM Cross Platform Component Object Model being developed by Mozilla, [Mozilla, 0].
The software on communications software can benefit from being structured in a modular way. Programmable network devices can also be structured by using component frameworks, and the software router presented in this dissertation also uses a component framework to structure its components and support the dynamic reconfiguration.

4.2.4 Computational Reflection

Computational Reflection [Mae, 87] is the process by which a program can access its internal structure and behaviour and also manipulate this internal structure therefore altering its behaviour. The mechanism by which a programmer performs a task using reflection involves the use of reification, which allows the programmer to inspect metadata within the program. Reflection has a potentially wide array of uses and as such has been used in a large number of research projects and commercial applications.

Most mainstream programming languages have a form of reflection available either inherently or more commonly bolted on as a separate API. For example since version 1.1 the Java Software Development Kit (SDK) has contained a reflection implementation [SUN, 98]. However this implementation can be considered "read-only" as a program can for-example query the methods of a class but can change the methods of the class.

A common way to use reflection other than within the programming language itself is by using a component model which has reflective abilities. One such modern component framework is OpenCOM [Coulson, 00] which is a COM [Microsoft, 02] based component model which allows the programmer to inspect the internals of a assembled system. This inspection takes the form of reifying the meta-interfaces of components and component frameworks; this accesses an internal system graph which maintains the state of the system. The programmer can also insert arbitrary code in bindings that is executed when a call is made across the binding. The model also contains a resources meta-model which represents types and quantities of various resources on the particular platform. OpenCOM is implemented in c++ and hence performs well. It has been used for a number of platforms including a reflective CORBA Object Request Broker (ORB) implementation called OpenORB [Coulson, 01].

Reflection can also be used as a foundation of other programming technologies, for example reflection is used in [Sullivan, 01] along with dynamism and metaobject protocols in order to provide the infrastructure needed to support aspect-oriented software development.

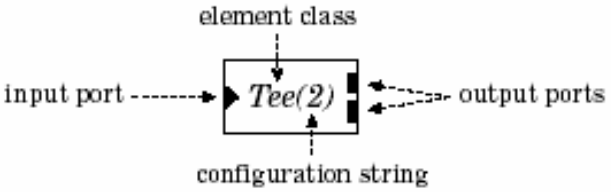
The remainder of this chapter presents a number of important and/or unique systems-level component frameworks, component models or programmable systems. Click, Router Plug-ins and LARA++ are specific frameworks which restrain the way in which the software is built to perform a task. THINK, OSKit and OpenCOM are component models for building systems software.

4.3 Click

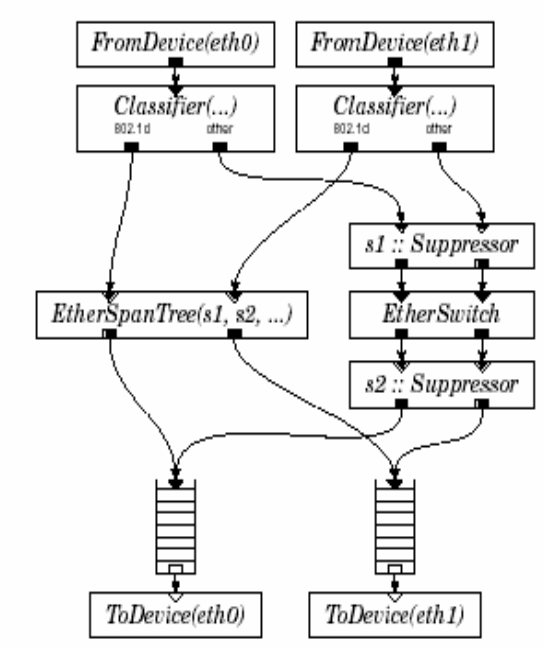
Click is a software architecture designed to enable the building of flexible and configurable routers. Click was developed at the Laboratory or Computer Science at MIT.

Click is a software architecture designed to enable the "building of flexible and configurable routers" [Karlin 01]. A click enabled router consists of a number of packet processing modules called elements. These individual elements implement simple router functions like packet classification, queuing, scheduling or network device access. For example, one such elements job would be to decrement an IP packets time-to-live field.

Figure 4.1 a click element



An element has a number of important properties. For example the element in figure 4.1 taken from [Karlin 01], has an "element class" which specifies the code that should be extracted when a packet is processed. It also has a number of ports; an output port on one element goes to a input put on another element, and each type of port can also have different semantics. A configuration string contains optional arguments that can be passed to the element at router initialisation time. Each element also has a number of method interfaces which allows elements to communicate at run-time.



Click configurations can take a number of different forms as they are modular and easy to extend. Different configurations considered include an Ethernet switch and an IP router, each with its own characteristics, like QoS or ECN.

A Click router configuration is a directed graph with elements at the vertices and with the packets themselves flowing along the edges of the graph. For example figure 4.2 (taken from [Karlin 01]) shows a click configuration for a simple Ethernet switch, the Ethernet devices are at the top and bottom, there are queues for the outgoing packet to the devices, the modules in the middle perform simple operations on the packets.

Figure 4.2 an example click configuration (from [Karlin 01])

Click itself runs as a kernel thread inside a Linux 2.2 kernel. The kernel thread runs the router driver itself which then loops over the task queue to run each task. However as there is generally only a single thread running each router there is little support or possibility of support for QoS in the Click system.

The Click framework includes several features that make individual elements more powerful and complex configurations easier to write. 'Pull connections' allow the modelling of packet flow which is driven by transmitting hardware devices. 'Flow-based router context' allows an element to locate other interesting elements.

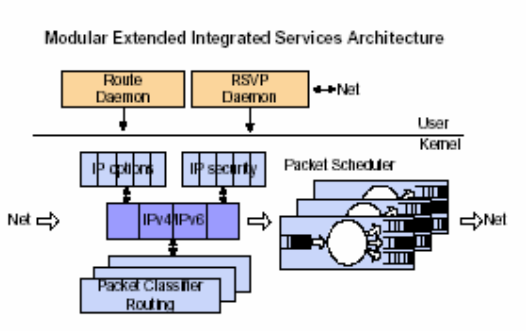
Analysis

A standards compliant IP router built using click has sixteen elements on its forwarding path, and using conventional PC hardware, this click IP router is capable of achieving a forwarding rate of 333,000 64-byte packets per second, indicating that such a structured architecture can achieve good performance. Although more support for QoS would be desirable and would lead to a more interesting and useful system.

4.4 Router Plug-ins

Router plug-ins is a high performance, modular, extended integrated services router software architecture for the NetBSD operating system kernel; it was developed at Eth Zurich in Switzerland and at the Applied Research Laboratory in Washington University,

Router plug-ins focuses on the increasingly important task of dynamically upgrading router software in an incremental fashion. The authors describe it as "a high performance, modular, extended integrated services router software architecture in the NetBSD operating system kernel" [Descasper, 98]. The architecture allows code modules called plug-ins to be dynamically added and configured at run-time. One of the novel features of the design is the ability to bind different plug-ins to individual flows.



The system is based upon an Extended Integrated Services Router (EISR) in figure 4.3 taken from [Descasper, 98], which features additional components to a conventional monolithic Best-Effort architecture. Including a packet scheduler, a packet classifier, security mechanisms, and QOS based routing.

Figure 4.3 Extended Integrated Services Router (from [Descasper, 98])

The goals of the Router Plug-ins framework are the implementation of algorithms into modules called plug-ins, extensibility; so that new plug-ins can be dynamically loaded at run-time, flexibility; in that instances of plug-ins can be bound to specific flows,

and importantly the system should provide for a very efficient data path, with no data copying, context switching or interrupt processing.

Figure 4.4, taken from [Descasper, 98], shows the system architecture and the control communication between different components. Here the IPv4/IPv6 consists of an implementation of IPv4/IPv6 which contains the few components required for packet processing which do not come in the form of dynamic modules. The plug-ins are visible at the bottom-right and include, in this case types for dealing with IP options, packet scheduling, packet classification and security.

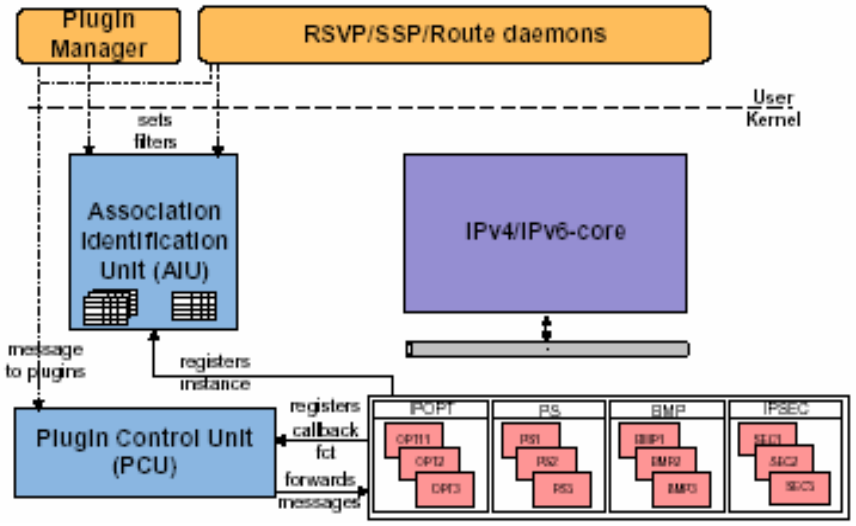


Figure 4.4 Router Plug-ins System Architecture and Control Communication ([Descasper, 98])

The plug-in control unit (PCU) manages and is responsible for the plug-ins. The Association Identification Unit (AIU) implements a packet classifier and "glues" the flows and plug-in instances together. The plug-in manager is a user space utility used to configure the system.

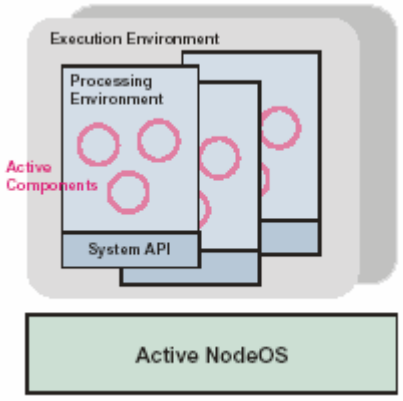
The primary goal of the architecture as the authors put it "was to build a modular and extensible networking subsystem that supported the concept of flows" [Descasper, 98]. The authors justify this because "the deployment of multimedia data sources and applications will produce longer lived packet streams with more packets per session than in common in today's environment". Also "in general integrated services router architectures should support the notion of flows and build upon it".

The authors' performance tests show that the system has an overhead above an unmodified NetBSD kernel of 8% and a slight decrease in packet throughput, with the advantage of a very modular design. These tests however show that on Integrated Services platforms, a very flexible and modular architecture can be introduced with almost no additional processing costs. Although Router Plugins is targeted at the PC environment, the system has been designed in such a way that the design would equiliy be relavent for more exotic platforms like Network Processors.

4.5 LARA++

The LARA++ active networking system [Schmid, 01] is a component-based system targeted at active network nodes. The LARA++ system aims to increase the flexibility of active network architectures by introducing a novel component-based active router architecture. The authors state that this will "provide a flexible and dynamically programmable platform for the development of active applications and network services based on the concept of services composition" [Schmid, 01].

The LARA++ architecture is based on layering active networking functionality on top of an existing router OS, which because the LARA++ architecture is general could be any supported OS, for example Linux on the IXP1200.



The architecture (illustrated in figure 4.5 taken from [Schmid, 01]) consists of; the NodeOS [NodeOS, 98] which provides low-level service routines and system policies, Execution Environments which form the management unit for resource access and security policies, Processing Environments which provide a safe and efficient execution environment for groups of active components and finally Active Components which comprise the actual 'active programs'.

Figure 4.5 Lara++ Architecture (from [Schmid, 01])

The NodeOS [NodeOS, 98] provides access to low-level system service routines and resources. It forms an independent but tightly integrated layer on top of the router OS. The Core components of the NodeOS are the packet Interceptor and injector (used in this project) which interface with the network stack. The packet classifier which determines the type of computation required. There is also a unique memory mapping mechanism designed to avoid expensive copy operations.

Lara++ components can be either active which perform active computation or passive which provide merely static functionality that can be used by the active components. They are implemented using Microsoft's COM component model so they are well understood and relatively simple to design, and utilise the well-known IUnknown interface to access component facilities.

Analysis

LARA++ components are different than what other systems in this document call a component. Most systems define a component to be pluggable modules of a system. A LARA++ component is a block of active network code which can be run on an execution environment in the LARA++ architecture.

The tests that have been carried out show that the use of LARAs processing environments introduces a sizable processing overhead, but that the system has the

benefits of the safe execution of potentially malicious active code. A number of optimizations have increased the efficiency of the system, including a user-level scheduler to reduce the performance hit associated with active scheduling.

The LARA++ System is targeted at a PC based environment running either Microsoft Windows or Linux, which don't have very strict requirements for resource or processor usage. Network Processors however have very stringent in its resource usage, so application development has to be much more carefully designed.

Furthermore the authors don't consider the issues of implementing the system on resource strict platforms like the Network Processors detailed in section 2. The system would however initially be compatible with the Intel and IBM platforms because they run a variant of UNIX, which the LARA++ system is compatible with. However it would need extensive modifications to support the specialised forwarding modifications of the platforms. For example, the comparable performance

However to sum-up, the design of LARA++ using a general architecture fits in well with the decision by most active router manufacturers to use a general CPU with some special packet processing characteristics, however this is not strictly true of Network Processor manufacturers where only a few use a central general CPU.

4.6 OpenCOM

OpenCOM is "a lightweight and efficient in-process component model developed at Lancaster Universities Computing Department, it is built atop a subset of Microsoft's COM" [Coulson, 00]. It also implicitly includes a mechanism for making the dependencies explicit between components. This mechanism is illustrated in figure 4.6 which shows an OpenCOM component (taken from [Coulson, 00]).

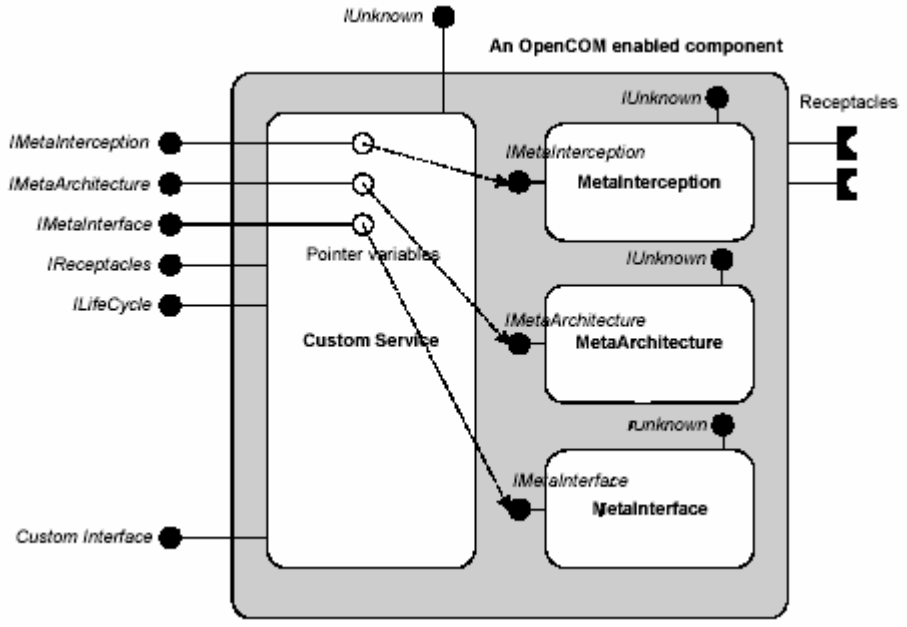


Figure 4.6 an OpenCOM component

The mechanism for explicit dependencies involves the use of receptacles. A receptacle specifies a interface which the component in question requires from another component before it can start. For example, if an OpenCOM component exposed receptacles of type A and B it would require components with interfaces of type A and B. This enables reconfiguration as when replacing a component or group of components the dependencies are explicitly encoded in the component so the component(s) will not be left without a requirement. OpenCOM also introduces "basic mechanism-level functionality for reconfiguration, including mutual exclusion locks to serialise modifications of inter-component connections" [Coulson, 00].

COM is a good basis for a lightweight component model because it is "standardised, well understood and widely used" [Coulson, 00]. It is language independent so it can be developed for a wide range of systems and it is also much more efficient than other component models like JavaBeans or the CORBA [OMG, 02] Component Model (CCM).

OpenCOM ignores "the higher-level features of COM such as distribution, persistence, security and transactions" [Coulson, 00], and utilities a number of more low-level features. OpenCOM only utilises COM's core features because the component models design is intended to aid the implementation of higher level features in the middleware environment constructed from components.

OpenCOM uses COM's binary-level interoperability standard which utilises the vtable data structure allowing function calls and object access. It also uses Microsoft's Interface Definition Language (IDL) allowing skeleton and stub creation, COM's globally unique discovery technique (known as GUIDs) and COM's IUnknown interface for interface discovery and reference counting. It also "adds support for Pre- and Post- method call interception which enables the injection of monitoring code" [Coulson, 00], and adds support for other reflective techniques.

The OpenCOM runtime manages the inter-component connections and the OpenCOM system itself. If a component is replaced it will insure that all the dependencies (receptacles) have been satisfied before starting it.

OpenCOM is still a highly experimental component model, it has however been used to build the OpenORB v2 which is a CORBA compatible component-based middleware platform. OpenORB is a general purpose middleware platform based on a set of flexible and configurable component frameworks (CF). It basically adds the notion of component frameworks to support the nesting of components onto the functionality provided by OpenCOM.

To support more platforms, OpenCOM is currently being extended to include support for pluggable loaders and binders. For each new platform that OpenCOM is targeted for a pluggable loader will be created which supports the initialisation and usage of components for that platform. A pluggable binder for a specific platform allows components within that platform to be binded together; a further binder extension will include support for binding components across platforms.

The designers note that as OpenORB is built on OpenCOM and that "receptacle and interceptor based invocations should not overly affect the performance of the system

then you would expect that OpenORB would perform on a par with a equally coded system without OpenCOM" [Coulson, 00]. Tests comparing OpenORB using raw RPC invocations show that it does performs on a par with Orbacus which is well known as one of the fastest CORBA compliant ORBS. And it was only slightly beaten by GOPI which is an experimental ORB build using C in UNIX and is what OpenORB was based upon, except GOPI is less flexible.

This therefore suggests that OpenCOM performs relatively will when pared with Component Frameworks for networking tasks, this makes a good case for using OpenCOM in other similar fields to middleware, including in this reports case, software for routers.

4.7 OSKit

The OSKit [Ford, 97] is a toolkit for developing Operating Systems developed at the Flux Research Group at the University of Utah. It consists of a configuration framework and a set of 34 component libraries oriented to operating systems, together with extensive documentation to aid in operating system development. The aim behind the development of the OSKit is to "make it vastly easier to create a new OS or port an existing OS to the x86 architecture". It also aims to allow the enhancement of Operating Systems to support a wider range of devices, file system formats, executable formats, or network services. The OSKit works well for constructing OS-related programs, such as boot loaders or OS-level servers atop a microkernel.

The OSKit uses a technique to allow unmodified code from existing mature operating systems to be included in a OSKit distribution. This is illustrated in figure 4.7, which shows the structure of the OSKit toolkit, including Native OSKit Code, Legacy Code encapsulated in OSKit modules and Glue code which wraps operating system dependent code.

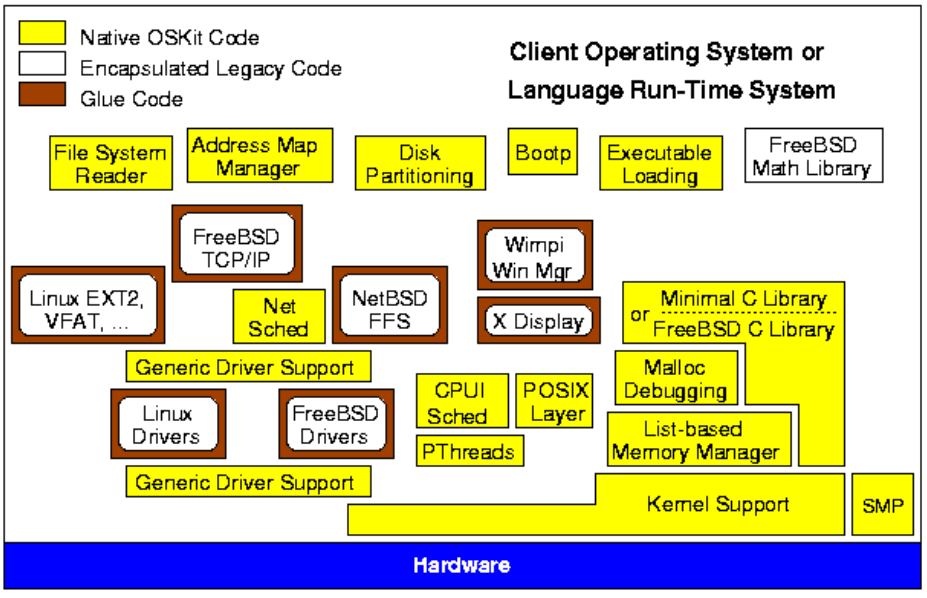


Figure 4.7 the structure of OSKit [Ford, 97]

OSKit contains support for core operating system functionality for supported architectures within its microkernel support. It then has a number of libraries which run on top of the microkernel to give extra support to high-level mechanisms. These libraries include memory management, scheduling and generic driver support. On top of these low level facilities there is support for legacy operating system code wrapped by 'glue' code. These facilities then allow more complex services to be built like, executable loading, disk partitioning and file system readers.

OSKit components are based upon a subset of the Component Object Model [Microsoft, 02]. OSKit uses COM as a framework in which to define its component interfaces, and as a language-independent protocol allowing software components within a address space to communicate with each other efficiently. Using COM to define OSKit's components allows them to retain suffucient seperate so they can be developed and evolved independently. OSKit maintains a degree of flexibility in component granularity by only specifying that a component should just referenced within a library by exposing its symbols. This means that any component code within a library is valid, with the obvious negative implications of security and code validity.

OSKit has successfully been used to implement a number of Operating systems, the most noticeable being and FSCK boot disk for FreeBSD partitions and a implementation of the Click Modular router. However, although the OSKit contains substantial machine-independent code, it currently only contains machine-dependent code for the Intel x86 and Digital DNARD (StrongArm SA-110 CPU). Although ports to other platforms are being considered, this is unlikely as development of the OSKit has slowed within the Flux Research Group.

4.8 THINK

THINK [Fassino, 02] is a software-framework for allowing OS designers to build a component-based OS out of arbitrary sizes of components. THINKs architecture is based on a number of simple concepts: components, interfaces, bindings, names and domains.

> Components and interfaces in THINK are similar to those in other component technologies. The authors cite ODP [ODP, 95] as the main source for these concepts, the difference being that in ODP they are called objects and interfaces.
>
> Names are used in THINK to give a context to a specific interface. They are also context dependent, which means that they relate to a certain context. A naming context is a set of names which have the same context, naming convention and name allocation policy. This is important as naming contexts can be organised in graphs by name, allowing dynamic replacement of components by name.
>
> In THINK, a domain is a set of interacting components binded together through their interfaces. Examples of domains cited by the authors are an

"operating system kernel executing in privileged processor mode" or a "set of user processes executing in unprivileged processor mode".

Components in THINK can be binded both statically and dynamically, and can even be created across domain boundaries. The THINK framework allows binding types to be pluggable in the same way that components are pluggable. A number of binding types have been implemented, including a remote communication binding and a system-call binding.

To aid programmers in using the THINK framework, a set of common operating system services have been developed and packaged as a library called KORTEX. As well as the KORTEX set of common libraries, THINK also has two tools to aid the programmer in the development of specialised kernels. An interface compiler is used to generate component code from Java specifications, for example C declarations and code that make up a component. THINK allows kernel images to be assembled by using an "off-line configuration" tool. This tool allows a programmer use code from the interface generator together with a UML specification of a system to generate an initial static configuration of the system. This initial configuration contains information about the bindings between components, as well as any information regarding inter-component dependencies. This configuration is then used to statically schedule component initialisation at boot-time.

The architects of THINK have used the framework to produce a number of example applications. These include a extensible distributed micro-kernel, a limited active networking kernel, a Java virtual machine and a version of Doom.

Analysis

THINK is similar to OSKit [Ford, 97], which allows the reuse of system components for Operating System Kernels. THINKs architects however argue that OSKit's components are too coarse grained for most applications, and that THINK's variable component size would be more suitable for certain applications such as those in the active networking domain. The only experiment undertaken in this field by the THINK team uses their KORTEX library with THINK to provide a base for a learning bridge protocol (Plearn) using the PlanP [] active networking language. This experiment is limited as they have only succeeded in showing that a THINK specialised kernel is more efficient than a Linux or Solaris generic one. They have also failed to illustrate the potential flexibility in the active networking field of the THINK architecture.

4.9 Analysis

A number of fields of component-based systems in computing are converging upon the programmable networking field. This chapter has presented technologies from a number of different fields all having the commonality of being systems-level component-based software.

A number of technologies have pursed the idea of dynamic bindings to support system level component platforms, these include OpenCOM and THINK. THINK has attempted to apply this to active networking with their PlanP experiments, and OpenCOM is looking to apply components generically to programmable networking with its current NETKIT project [Coulson, 02].

In relation to Network Processors, the field of Systems-level component software is mainly using the facilities of Network Processors because of their flexibility in order to prove the various aims of the software. NetBind and VERA go some way to provide facilities for Network Processors, but this is done without the direct aim of improving the software for Network Processors.

The main problem identified with software collected in this chapter is that the more support a platform gives to implementing a specific feature like dynamic-reconfiguration, that the closer that software becomes tied to a specific technology and hence less generic it becomes. This often limits the viability of such software as it can only be adopted on a small subset of hardware platforms which are quickly outdated. This also hinders any possibility of network/system wide deployment and management. So fundamental in the design of systems-level software if it is to be generic and viable is to design the component model and the accompanying component frameworks to be as generic as possible.

5 Research Proposal

5.1 Introduction

This Chapter presents a proposal of the research to be carried out in the remainder of the PhD. It presents the mains aims, specific objectives, methodology and plan of work to be carried out.

5.2 Statement of aims

This section presents the general aims of the proposed research together with the required goals; it then details the main concrete objectives which will be used to evaluate the work.

5.2.1 Main aim

Current commercial software and academic research aimed at Network Processors fail to provide programmers and system designers software techniques to support the creation of optimized software for different Network Processors. Furthermore all commercial software and most academic research fail to address the issue of supporting runtime reconfiguration which is emerging as an important requirement of any network device (see section 2). Those research projects like NetBind and LARA++ which focus on supporting high-flexibility and runtime reconfiguration fail to perform as well as optimized non-flexible solutions.

Essentially the problem is that none of the current available research succeeds in enabling the development of software with design portability for Network Processors. And current software fails enabling runtime reconfiguration in a uniform and optimized way across multiple Network Processor platforms. In addition, for any research with this aim to be accepted it must also have the following characteristics:

1. A single development environment for a number of platforms and a uniform set of deployment, re-configuration and analysis tools. Any development software targeted at Network Processors must be able to show that it is scalable, and demonstrate that it could apply to other platforms.

2. Support for runtime "dynamic" reconfiguration, which enables administrators or user applications to alter the internal state of the Network Processor software; this must have as low an overhead as possible to reduce cost due to potential downtime.

3. A number of Component Frameworks for different application scenarios in order to prove the thesis. This will take the form of libraries of components to aid in efficient development.

[Gottlieb, 02] demonstrates that a simple model can be used to describe the general operation of an extensible router; to classify packets, process them and schedule them for transmission. Whilst [Gottlieb, 02] concentrates on three "traditional" extensible routing architectures (Scout [Mosberger, 96], Click [Kohler, 00] and Router Plugins [Descasper, 98]) the proposed project aims to expand the theory that a simple model can describe the general operation of extensible routers to encompass network processors.

This task as proposed will involve the use of a component model with a runtime running on the host processor of the chosen platform, which for the first stages of the project will be the Intel IXP1200. The project will utilise our dynamically reconfigurable component object model called OpenCOM [Coulson, 00]. This is an ideal component model to be used in this situation as it is lightweight, with minimal overhead at runtime and has been used successfully and efficiently in a number of middleware platforms [Coulson, 00].

5.2.2 Specific Objectives

In order to meet the primary aims the following specific sub goals must be achieved:

1. A key sub goal is to design and implement a reflective, component based infrastructure suitable for Network Processors that allows dynamic manipulation of the platform. It must be lightweight in nature to overcome the limited resources of network processor platforms, and it must perform well as performance is one of the main goals of network processors in general. For the initial stages of the project the component platform will be developed for the Strong-Arm processor present on the Intel IXP1200 Network Processor. This will involve the adaptation of our current *NIX platform to deal with the intricacies of the platform as well as improvement in certain areas including performance and programmability.

2. The second key goal is to design and implement the mechanisms which will allow the platform to manipulate network processor intricacies like the microengines, different memory types, coprocessors and hardware assist functionality. This will be done by implementing facilities in OpenCOM which will allow components to be developed, deployed and managed for Network Processors. Thus network processor facilities will be wrapped as OpenCOM components or as binding types in OpenCOM. These two key sub-goals will render component-based dynamically reconfigurable platform tailed for the IXP1200 Network Processor.

3. For any development platform to be successful or in fact even usable it must include a number of fundamental features which enable the application programmer to produce viable software. So as well as producing a viable component model (sub-goal 1) and support for the specific board facilities (sub-goal 2) the aim must be to develop Component Frameworks and toolsets to support the platform. The Component Frameworks will be developed with the aim of simplifying the programming of the intricacies of Network Processors, which is currently very difficult. It will attempt to hide the complexity whilst still allowing powerful fine-grained components which

manipulate the platform heavily to be developed. The toolset will consist fundamentally of tools for compiling and binding components as well as deploying and configuring the runtime both locally and remotely. The reality of the toolset will be focused on using the current available tools provided by Intel and tailoring them for this target, most of the development will be focused towards run-time tools. As the platform matures, the later stages of the platform development may contain graphical component integrators, component generators as well as analysis tools, this further work could be carried out by a MSc student as part of their research project. These 3 sub-goals will complete a useable platform for the Intel IXP1200 similar to our current WIN32 OpenCOM version.

4. Central to the research proposal is the objective to produce a "generic" component model which can be used to ease and improve the development of software for multiple Network Processors. Hence a major body of the work must be an attempt to prove that the platform for the IXP1200 Network Processor can be adapted to support different Network Processors. This would be carried out in the form of a detailed research and design exercise involving a number of network processors. This goal would be undertaken concurrently with goals 1-3 to attempt to create generic platform which would be viable on multiple and widely varying platforms. This goal has the ultimate goal of providing the application programmer with a generic interface to develop for both platforms with the same API and component model and hence proving the thesis of this research proposal.

The first two objectives illustrate the key goal of the project to create a component-based platform for Network processors. The third objective focuses on extending the component-model to become a development platform with the necessary tools to accomplish that goal. The fourth objective focuses on the main goal of the project to produce a generic platform for network processors., and aids in the width and depth of the final research thesis. However, in order to meet these goals and add further value to the project, the following objectives have been identified:

- During development of all stages of the project it is obvious that as well as the platform and toolsets there is also a need for example applications to be developed to evaluate the power of the platform. This will initially consist of a L3 forwarder based on the RFC 791 standard allowing IP packets to be routed. This can be categorised for work-unit purposes as a port of current L3 forwarder implementations for the IXP1200, for the Intel ACE platform or for NetBIND. This will serve the basis for performance tests which will attempt to prove that the platform is a viable and feature rich alternative to traditional platforms.

- As discussed in this report, support for dynamic reconfiguration is a desirable characteristic for any networking platform. Therefore a distinct goal is to develop the reflective abilities of the platform to support network processors. The L3 forwarder developed will be useful in developing the dynamic reconfiguration mechanisms as there is definitive possibilities for the L3 forwarder to be modified at run time. For example the L3 forwarder could be

augmented with a diffserv implementation or with round-robin queue management or support for forward error detection.

- The goal of the proposed project is to prove such a platform as described in this proposal can be generic, so an important part of the project will be to study the trends in network processor design. The investigation will focus on the technical detail of architecture/programming and reconfiguration potential of current network processor designs. There are considerable problems with designing a platform to be generic especially in fields of relative infancy like that of network processors. As can be seen from chapter 3 the variety of network processors is large, with some platforms having a central processor which would support our platform and some which don't have a central processor which would imply a complete redesign. Hence our survey would concentrate first on a subset of network processors with a central control processor like the Intel's Strong-ARM in the IXP1200 or the IBM PowerPC in the PowerNP. To prove that the proposed project is viable across widely varying platforms, the study will also be undertaken on other "different" Network Processors like those discussed in sections 2.5-2.8.

These final objectives are not central to the goal of the project, but rather they act to support the key issues.

5.3 Methodology and approach

The methodology that will be adopted to complete this project will follow an incremental approach. The initial component model will be implemented on the Strong-ARM processor of the Intel IXP1200; it will then be tested to guarantee it meets the required characteristics and performance levels. Furthermore during sub-goal 2 and 3 the platform will be ensured to be suitable for real network applications by developing example applications alongside the platform itself. For example this could take the form of IP L3 forwarder, which would be developed first statically and then with dynamic reconfiguration support. The use of pluggable loaders and binders (see section 4.6) will greatly ease the task of integrating the current OpenCOM work and its code base with the newly abstracted and heterogeneous components for the ixp1200. As the platform matures, this application could be expanded and used to testing and performance measurements. Furthermore having a viable application of the platform makes applying the thesis of a generic component model for network processors easier to test in the early stages.

Following on from the successful implementation and testing of the initial IXP1200 prototype, the focus will move to proving that the prototype could be applied to other Network Processors. This will take the form of a detailed theoretical and technical study into other Network Processors.

5.4 Programme of Work

This section outlines the main tasks and sub-tasks required to meet the aims of the project. A timetable describing the completion of these tasks is given in figure 7.1. The work described begins at the start of the second year of this PhD (October 2003) and describes the development process over the following 18 months.
The following list describes the individual tasks which must be completed in order for the project to be a success. The list describes what the task involves, the related deliverable and the length of time to complete the task. At the end of the section is a timeline which shows the relationship between tasks.

Task 1: Continuing Study of Network Processor technology
Deliverable: Continuing investigation into the various network processor platforms available. This will include detailed technical and architectural studies and viability studies relating to the viability of the project goals.
Time: 18+ months

Task 2: Design and implementation of the underlying component model for the Network Processor
Deliverable: The architectural design and documentation for the platform, which will be capable of managing components dynamically. The implemented prototype will be capable of loading, unloading, binding and unbinding components on the Strong-ARM processor. This will include changes to the component model runtime to support this.
Time: 3 months

Task 3: Iterative design and development of mechanisms for network processor specific component frameworks.
Deliverable: The component model and component frameworks include support for the network processor specific mechanisms.
Time: 3 months

Task 4: Testing of the IXP1200 prototype
Deliverable: The test results for the prototype, which will ensure that the correct level of functionality and performance by the platform are met. These tests will be carried out alongside the development of the application in task 5 and 6.
Time: 5 month

Task 5: Design and implementation of a L3 forwarder application
Deliverable: A design of an L3 forwarder for the IXP1200 prototype platform taking into account the characteristics of the platform. An implementation of the design which is optimized for the Intel IXP1200 network processor and its specialist characteristics.
Time: 4 month

Task 6: Network Processor Studies
Deliverable: Detailed technical study of the viability of applying the design applied to the IXP1200 to another Network Processors. This complements the ongoing general investigation into network processors in task 1.
Time: 3 months

Task 7: Evaluation of work
Deliverable: Detailed technical evaluation and of all the work undertaken during tasks. This will consist of performance tests as well as qualitative evaluations. The work will be compared with similar research to evaluate its validity. The result will be a rigorous evaluation used for assessing the usefulness of the prototype software and assessing the thesis of the research itself.
Time: 3 months

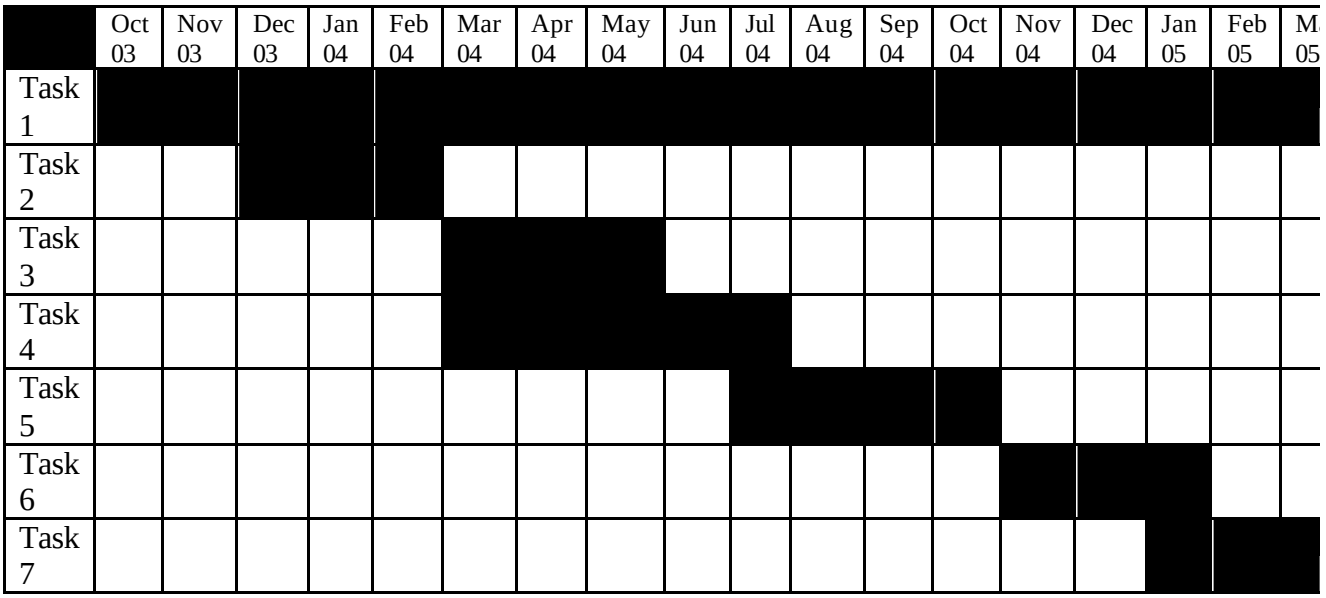


Figure 5.1 Programme of work

6 Conclusions

This document has presented the state of the art in the field of network processors and the software needed to utilise them as efficient programmable networking platforms. It has presented the problems of lack of programmability and dynamic adaptation in current network processor platforms and the effect this has on the effectiveness of the platforms. In addition following the theme of increasing the programmability of networking platforms a number of systems-level component software platforms have been reviewed.

We theorise that utilising the power of a reflective component model will support the need for dynamic reconfiguration in Network Processors and that this can be done in an efficient manner. We hope to do prove that this thesis is viable over the wide spectrum of Network Processing platforms.

References

[Agere, 03] Agere Systems, "Advanced PayloadPlus® Network Processor Catalog" Agere press, 2003

[Baker, 95] Baker F., "Requirements for IP Version 4 Routers: RFC 1812", Internet Request for Comments, June 1995.

[Campbell, 01] Campbell A., Chou S., Kounavis M., et al, "NetBind: A Binding Tool for Constructing Data Paths in Network Processor-Based Routers", XXX

[Cisco 03] Cisco systems"Parallel Express Forwarding on the Cisco 10000 Series" Cisco Systems, Inc. 2003

[Comer, 03] Comer D E., "Network Systems Design using Network Processors", IXP edition, Prentice Hall Press, 1st edition 2003

[Coulson, 00] Coulson G., Blair G S., Clarke M., Parlavantzas N., "An Efficient Component Model for the Construction of Adaptive Middleware"., DMRG, Computing Department, Lancaster University and Dept of Computer Science, Tromso University, 2000.

[Coulson, 01] Coulson G., Blair G S., Clarke M., Parlavantzas N., "The Design of a Configurable and reconfigurable middleware platform", DMRG, Computing Department, Lancaster University and Dept of Computer Science, Tromso University, 2000.

[Coulson, 02] Coulson, G., Blair, G. Hutchison, D., Ueyama, J., Irvin, Y., Lee, K., Surajabli, B., "NETKIT: A Software Component-Based Approach to Programmable Networking", Lancaster University, 2002.

[Descasper, 98] Descasper D., Ditta Z., Parular G., Plattner B., "Router Plugins, A Software Architecture for Next Generation Routers", Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland, and Applied Research Laboratory, Washington University, Proceedings of the ACM SIGCOMM, Sept 1998.

[EZchip, 03] "Network Processor Designs for Next-Generation Networking Equipment White Paper", EZchip technologies.

[Fassino, 02] Jean-Philippe Fassino, Jean-Bernard Stefani, Julia Lawall, Gilles Muller, "THINK: A Software Framework for Component-based Operating System Kernels", appeared in USENIX'02

[Ford, 97] B. Ford, G. Back, G. Benson, J. Lepreau, A. Lin, and O. Shivers. The Flux OSKit: A substrate for kernel and language research. In SOSP'97 [29], pages 38–51.

[Gottlieb, 02] Yitzchak Gottlieb and Larry Peterson. "A Comparative Study of Extensible Routers," in 2002 IEEE Open Architectures and Network Programming Proceedings, 51-62, June 2002.

[IBM, 02] IBM,"IBM PowerNP Software Offerings", IBM, September 2002

[IBM, 03a] Björn Liljeqvist , "IBM Power NP4GS3", from "A Survey of Network Processors" at http://tech.nplogic.com/, 03

[IBM, 03b] James Allen, Brian Bass, Claude Basso et al., "IBM PowerNP network processor: Hardware, software, and applications", IBM Research press.

[InfoWorld, 02] Stephen Lawson, "Analysis: Network processors enter new generation", Infoworld (www.infoworld.com), June 19th 2002.

[Intel, 97] Intel Corporation, "Gigabit Ethernet", Intel Networking Technical Briefs, Intel Corporation, 1997.

[Intel, 00] Intel Corporation, IXP1200 Network Processor Datasheet, Sept 2000

[Intel, 03] Intel Corporation, Internet Exchange Architecture, http://www.intel.com/design/network/ixa.htm, 2003

[Karlin 01] Karlin S., Peterson L., "VERA: An Extensible Router Architecture", Department of Computer Science, Princeton University, 2001.

[Kohler, 00] Kohler E., Morris R., Chen J., Jannotti J., Kaashoek M F., "The Click modular router", ACM Transactions on Computer Systems, 18(3):263-297, Aug 2000

[Ford, 97] The Flux OSKit: A Substrate for Kernel and Language Research (1997), Bryan Ford Godmar Back Greg Benson Jay Lepreau Albert Lin Olin Shivers University of Utah University of California, Davis Massachusetts Institute of Technology

[Kounavis, 03a] Kounavis M., Campbell A., Chou S., et al, "A Programming Environment for Network Processors", Network Processor Conference, West, San Jose, CA, October 2003.

[Kounavis, 03v] Kounavis M., Kumar A., et al, "Directions in Packet Classification For Network Processors, *Second Workshop on Network Processors*, Anaheim, CA, February 2003.

[Oracle, 03] Oracle Corporation, http://www.oracle.com, 2003

[Mae, 87] Maes P., "Concepts and experiments in computational reflection", proceedings of the 2nd annual conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA) June 1997

[Microsoft, 02] COM, Microsoft Corporation, http://www.microsoft.com/com/, 2002.

[Mosberger, 96] Mosberger D., Peterson L., "Making paths explicit in the scout operating system", Proceedings of the second USENIX Symposium o Operating Systems Design and Implementation, pages 153-167, Seattle, WA USA, Oct. 1996

[Motorola, 03] Motorola group, C-5 Network Processor Data Sheet Silicon Revision D0, Motorola press, 2003

[Mozilla, 01] Mozilla Organization, XPCOM project, 2001, http://www.mozilla.org/projects/xpcom.

[NetSpeed, 03] "Programming Challenges", http://www.network-speed.com/Technology/prog-challenge.html, Network Speed Technologies Inc, 2003.

[NodeOS, 98] Architectural Framework for active Networks, DARPA AN working group Draft, 1998.

[NPForum, 03], Network Processor Forum, website www.npforum.org.

[ODP, 95] ITU-T Recommendation X.903 — ISO/IEC International, Standard 10746-3. ODP Reference Model: Architecture. ITU-T — ISO/IEC, 1995.

[OMG, 02], CORBA, www.corba.org, the object management group, 2002.

[Real, 03] Real Networks, www.real.com

[RFC791] "Internet Protocol", Request for Comments, RFC791.

[RFC1889] "RFC 1889 - RTP: A Transport Protocol for Real-Time Applications", Request for Comments, RFC1889.

[RFC2460] "Internet Protocol, Version 6 (IPv6) Specification, Request for Comments, RFC2460.

[RFC2815] "RFC 2815 - Integrated Service Mappings on IEEE 802 Networks", Request for Comments, RFC2815.

[RFC3086], "Definition of Differentiated Services Per Domain Behaviors and Rules for their Specification", Request for Comments, RFC3086.

[S3, 03] Silicon & Software Systems, www.s3group.com/network_processing/

[Schmid, 01] S. Schmid, J. Finney, A.C. Scott, W.D. Shepherd, "Component-based Active Network Architecture", In Proc. of 6th IEEE Symposium on Computers and Communications (ISCC), Tunisia, July, 2001.

[Schmid, 02] Schmid S., "A Component-based Active Router Architecture, PhD thesis, Distributed Multimedia Research Group, Lancaster University 2002

[Spalink, 01] Tammo Spalink, Scott Karlin, Larry Peterson, Yitzchak Gottlieb, "Building a Robust Software-Based Router Using Network Processors", In

Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP'01) . pages 216--229, Oct 2001

[Szyperski, 98] Szyperski, C., "Component Software: Beyond Object-Oriented programming", Addison-Wesley, 1998.

[Sullivan, 01] Sullican G., "Aspect Oriented Progamming using Reflection", Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Appeared in OOPSLA workshop on Advanced separation of concerns in Object-Oriented Systems 2001

[Sun, 98] "Java core reflection", Sun Microsystems, 1998.

[Teja, 03] Teja Technologies, www.teja.com

[Thinkofit, 03] Thinkofit, "Real-Time Conferencing, Guide to Internet Conferencing", http://www.thinkofit.com/webconf/realtime.htm, 2003

[XBOX, 03] XBOX Live, Microsoft Corporation, http://www.xbox.com/uk/live, 2003