The Suitability of Network Processors For Multi-field Packet Classification

Prepared by:
Tyrell Sassen

Supervised by:
Neco Ventura

Department of Electrical Engineering
University of Cape Town
2006

This dissertation is submitted to the University of Cape Town
in fulfilment of the academic requirements
for the Degree of Master of Science in Engineering

20 January 2006

Declaration

I declare that this thesis is my own work. Where collaboration with other people has taken place, or material generated by other researchers is included, the parties and/or material are indicated in the acknowledgements or references as appropriate.

This work is being submitted for the Master of Science Degree in Electrical Engineering at the University of Cape Town. It has not been submitted to any other university for any other degree or examination.

Signed by candidate

Tyrell Sassen

20/01/2021

Date

Acknowledgements

I would like to thank the following individuals and organisations for their assistance and guidance:

- Neco Ventura, for supervising this research.
- Telkom, Intel, Siemens and the Department of Trade and Industry (DTI) for their generous sponsorship towards this work.
- My colleagues at the Communications Research Group at UCT, for their technical support and feedback.

Synopsis

As the Internet continues to grow, there is an increasing need for intelligent network elements that can adapt their behaviour to fit the needs of a particular network application. In order to do this, the network elements must be able to differentiate between traffic streams. This is done by classifying packets based on multiple fields within the packet header and body.

Packet classification is a complex task, with poor worst-case bounds. Therefore current algorithms use a heuristics-based approach to simplify the problem. These algorithms use a variety of different mechanisms and a flexible, yet powerful architecture, is needed to implement these.

Network processors meet these requirements by using a software-programmable, parallelised design to process packets at line speeds. Multiple packet processing engines, operating in parallel, allow multiple packets to be processed simultaneously, while fast external memory is used to store the data-structures needed for packet classification. The algorithms themselves, were mapped to the MEs with only minor alterations. The structural model of the processor used a number of pipelines, which classified packets in parallel. A parallel/pipelined approach was used to allow the processing elements of the network processor to be assigned proportionally, based on the complexity of the task.

The evaluation framework was modelled on the DiffServ architecture and used a number of rule-sets, resembling real-world classification databases, to test the packet throughput rates and storage requirements of each algorithm. It was found that the performance of each algorithm varied with the size and structure of the rule-sets, though the maximum throughput was limited by the memory bandwidth of the network processor. The packet classification algorithms which provided the greatest throughput were those with the fewest number of memory accesses per packet.

Overall, it was shown that it is possible to classify packets at realistic network line-speeds, while still remaining within the memory limitations of the network processor.

Contents

List of Figures

List of Tables

Abbreviations

ACL – Access-Control List

AF – Assured Forwarding

ASIC – Application Specific Integrated Circuit

BA – Behaviour Aggregate

BV – Bit-Vector

CAM – Content-Addressable Memory

DSCP – Differentiated Services Code-Point

EF – Expedited Forwarding

E-TCAM – Extended TCAM

FPGA – Field-Programmable Gate Array

GPP – General Purpose Processor

IXA – Internet Exchange Architecture

MAC – Media Access Control

ME – MicroEngine

MSF – Media and Switching Fabric

NN – Next-Neighbour

NP – Network Processor

PHB – Per-Hop Behaviour

PPE – Packet Processing Engine

QoS – Quality of Service

RFC – Recursive Flow Classification

RSVP – Resource-Reservation Protocol

SLA – Service Level Agreement

TCAM – Ternary Content Addressable Memory

VPN – Virtual Private Network

Chapter 1

Introduction

1.1 Background Information

The origins of the Internet are well known, yet the large international network we know today is very different from the research network once known as ARPANET [47]. The idea was to create a network that functioned in a packet-switching manner, rather than the traditional circuit-switched telephone networks of the time. In the 1980s the U.S. Department of Defence stopped funding the ARPANET programme and the responsibility of maintaining the long-haul backbone of the Internet was handed over to the National Science Foundation (NSF). The resulting NSFNET maintained ruling restricting the use of the Internet for commercial purposes until 1994, when the backbone of the NSFNET was replaced by those backbones created by various commercial Internet Service Provider (ISP) companies [1]. This allowed for the commercial use of the Internet and various companies started to take advantage of these new opportunities.

For companies, the Internet has not only provided another means of advertising but it has allowed them to move outside the bounds of their physical buildings and into the comfort of the customers' own homes. This has led to the creation of "Non-store Retailers" – companies that do not have a physical retail presence but operate solely on the Internet. In fact, this industry has grown so large that in 2003 online retailers generated $56 billion in sales in the U.S. alone [11]. This figure represents 1.7 percent of total retail sales in 2003 and has grown by 25 percent since 2002 – this compared to a total retail sales growth of only 4 percent.

This increased level of Internet-commerce, together with the increased accessibility to the Internet, has created an arena with a wide-range of security threats. In 2004, the Computer Emergency Response Team Coordination Center (CERT/CC) published a survey [9] reporting that 70 percent of the companies questioned had experienced electronic crime committed against their organisation, which cost their respective companies approximately $666 million in 2003. These concerns have led companies to implement a number of different security mechanisms in order to protect themselves from these malicious attacks - this could be anything from a Virtual Private Network (VPN), used to connect internal networks across a large public network like the Internet, to simple mechanisms like firewalls, which protect an internal network from malicious activity. Although these mechanisms vary in design and location on the network, they all contain similar functionality. A traffic stream must firstly be examined to determine if it is malicious or not. Once this has been determined, the security mechanism can use this information to decide the correct procedure that must be taken regarding the traffic.

Besides the need for security, the fast commercial growth of the Internet has also sparked a number of innovative services. Some of the earliest uses of the Internet were applications such as email [43], file transfer [42] and news transfer [28]. All of these early applications are said to be non real-time applications, in that they do not require strict time-delay bounds in order to operate correctly. In contrast to this, recent trends have seen Internet applications becoming more real-time orientated. Applications such as Voice over IP (VoIP) require certain minimum bounds to be met in order to provide an acceptable level of service [68, 26]. The Chairman of the Federal Communications Commission (FCC) in the U.S.A. recently stated that the number of U.S. firms using VoIP is expected to grow to 19 percent by 2007 [37]. He also stated that 73 percent of wire-line service providers have either implemented, or were testing, VoIP in their networks.

Even though the TCP/IP protocols have maintained a great deal of popularity since their inception in 1983, so much in fact that they have become the protocols of choice in the converged Next-Generation Network (NGN), they are not without their shortcomings. The Internet was originally designed as a packet-switched network, and although this design allows more flexibility in terms of link sharing, they also have several disadvantages over circuit-switched networks. In a circuit-switched environment, the network must provision all resources required to create a connection before the circuit can be established. This can be best seen in the early stages of the telephone network, where, in order to make an international telephone call, a user would request a link from the telephone company for a

specific time and duration, before the call could be made. This provisioning of resources ensured that there was exactly enough resources available to place the call. In a packet-switched environment this is not the case; because the link is being shared, the more traffic that appears on the link the less resources (e.g. bandwidth) can be given to each individual connection. IP was created to be a best-effort protocol, where the best effort is made to deliver the data, though no guarantees are made regarding resources being available to correctly deliver the data in a reasonable amount of time. TCP builds on top of IP to provide a certain level of guarantee, where packets are retransmitted should they become lost or corrupt, however it still does not provide a mechanism for ensuring the timely delivery of the data. This best-effort model leads to problems when trying to support real-time services. Since IP does not provide any traffic differentiation, but instead treats all traffic equally – giving each stream in a shared link the same level of service, bandwidth sensitive application like VoIP can be adversely effected when link utilisation is high.

Different applications will have different requirements with regards to the service that they require. There are a number of different parameters that define the level of service that the network can achieve [38], these include network availability, bandwidth availability, delay, packet loss and jitter. Depending on its particular needs, an applications will place different priorities and bounds on these parameters to make up what is known as the Quality of Service (QoS) requirements for the application.

In order to support real-time applications over packet-switched networks, it becomes necessary for the network to provide bounds on the QoS provided by the network. Obviously, different services have different QoS requirements and it is in the interest of the network to be able to provide these different levels of services in order to maximise the usage of the network. In the past, the only way to guarantee that applications would receive the QoS bounds that they required was to over-provision the network. This allowed the network to operate at a low utilisation therefore guaranteeing that traffic would never be dropped on account of congestion. For this to be the case, the core network must provide bandwidth greater than or equal to the sum of the peak bandwidths required by each ingress link. This is inefficient as studies have shown that current link utilisation is only in the regions of 10-15% – with peak utilisation of approximately 45% [40].

The introduction of QoS schemes allows network operators to have higher-utilised core networks, while still being able to guarantee a constant level of service to network applications. This is done using a packet scheduling and buffer management techniques that allow certain higher-requirement traffic to be given preference over other lower-requirement

traffic. There have been various proposals to extend the TCP/IP protocol suite to provide QoS and service differentiation, the most popular of these being the *Integrated Services* (IntServ) [7] and *Differentiated Services* (DiffServ) [6] architectures.

DiffServ has risen as the favourite of these two architectures, due to the simplicity of its design. Unlike IntServ, which requires network nodes to store state information regarding the traffic flows being forwarded though the router, DiffServ keeps to the philosophy of implementing the complexity at the edge of the network, while keeping the core network simple and therefore fast.

Along with the increased requirement for QoS in packet-switched networks, the other major trend is the increased speed of the core and access networks. The initial proposal for ARPANET used modems operating over leased lines to provide 50 Kbps backbone connections [47], nowadays however, it is not uncommon to see backbone links in the 9.6 Gbps (OC-192) range [40]. This large increase in bandwidth is not limited to the core network, but can be seen right the way to the access network, where 5 years ago users connected to the Internet using 56 Kbps analogue modems, technology has advanced to the point where ISP companies like Verizon are offering 4.0 Mbps ADSL connections over standard telephone lines as well as speeds of up to 30 Mbps over fibre cables to the home [67]. This increase in access speeds, along with the increased growth of Internet users, has led to the need for more powerful network elements, to deal with the processing of this extra traffic.

While traditional IP forwarding can be done at relatively high speeds, providing additional services (like QoS provisioning) require a great deal more processing power. In the past ASICs were the only devices that could provide such functionality, with the performance required to operate at line speeds. This is changing as ASICs lose favour as network elements due to their long design cycles and inflexibility. *Network Processors* (NP) have moved in as a relatively new element for this market, their software-programmable architecture and high processing speeds – due largely to highly parallelised design and dedicated hardware units – make them ideal for complex tasks like layer 4-7 packet processing [49]. One of the biggest advantages of NPs (over ASICs) is their flexibility. Since the functionality of the NPs is implemented in software, it becomes possible to update the devices with little more than a firmware upgrade. This is especially beneficial in the QoS arena, as standards are still being developed and refined.

The network in Figure 1.1 gives a few examples of where QoS and network security would

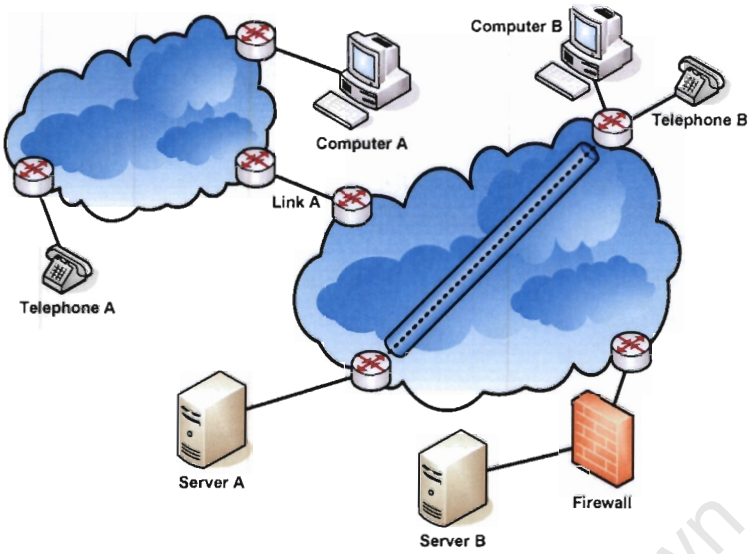

Figure 1.1: An example network showing various QoS requirements

be necessary in a typical network environment.

- Computer A wishes to download a file from Server B, Server B is situated in the company's private network, but is still accessible to the outside world. The firewall that lies between Server B and the public network, protects the server from malicious attacks by restricting the IP addresses that are allowed to connect to Server B. The firewall also restricts access to the TCP ports that a client can connect to.

- The user of Telephone A wishes to have a conversation with the user of Telephone B. These telephones use VoIP to allow the users to talk to each other since they are communicating over an IP network. The network provides QoS capabilities in order to ensure that the telephone call receives an acceptable level of service. Link A uses a QoS mechanism to provide a minimum amount of bandwidth and ensure a minimum amount of delay for VoIP calls between the two sub-networks.

- Computer B needs to connect to Server A in his remote office. The user connects to a VPN that allows him to be connected to the office internal network securely, regardless of his physical location. The VPN that he has connected to, has been configured using a QoS scheme ensuring it behaves like a leased-line with regards to bandwidth and other QoS parameters.

The above scenarios are typical cases in which QoS and security schemes will need to be implemented. These schemes allow a number of different services to operate on the shared Internet, while still providing service level bounds that give the illusion that they are operating on dedicated links. While the technology needed to make this example a reality already exists, it has not been implemented throughout the Internet. While QoS mechanisms are often implemented on links between commercial networks, there does not yet exist a universal QoS implementation that can offer guaranteed service levels to the endpoints of a user connection.

Although, both QoS and network security seem like wholly different issues, they both contain a common, initial step: packet classification. Before a data stream can be given a specific level of service, the stream must be identified as requiring a specific level of service. It is often impractical to let the stream decide for itself what level of service it requires. as this is open to abuse by malicious users. We must therefore have some network element that will classify the data as belonging to a particular service class, requiring a particular level of service. The same can be said in terms of network security, in order to identify network traffic that is of a malicious nature, the traffic must first be classified to determine if it fits the pattern of malicious traffic. Once the traffic has been classified in this way, it can be treated in a specific manner.

Traffic is usually classified based on the fields found in the packet headers. For instance, a TCP destination port of 80 will most often identify HTTP traffic, a classifier could use this information to give HTTP-request packets a higher priority than other miscellaneous traffic. While current packet classifiers rely mainly on the TCP/IP headers of a packet, it is becoming more common for network elements to perform what is known as "deep packet" classification. This involves examining fields from the body of a TCP packet - including application-layer fields. Obviously, the more headers that need to be matched, the more complex the operation. This rise in complexity has led to the wide-spread use of network processors to support these kinds of applications [35, 25, 10].

1.2 Thesis Objectives

This study intends to investigate the plausibility of performing packet classification on network processors. Packet classification plays an important part in providing a number of different IP services, including network security and QoS. Although it does not provide

the main functionality of these applications, it is often the first step that must be taken when traffic arrives at a network device. Due to this, a fair amount of research has gone into proposing packet classification algorithms that can provide better performance [16]. Some of the metrics that are used to determine performance include storage requirements, time complexity, preprocessing time and update complexity[1]. The focus of this study will therefore lie in comparing these existing algorithms with regards to their storage and time complexity, in order to determine which will be best suited for implementation on the network processor.

In the first stage of the study, a theoretical analysis will be performed on the different packet classification algorithms. This will help identify the algorithms which would be best suited for implementation on the NP. The algorithms will be grouped into different classes, with regards to relating characteristics. Once these classes have been identified, the most suitable algorithm will be chosen from each class for further testing. These implementations will then be tested against each other, with a number of different sized classification rule-sets. From these tests, the best algorithm will be identified; though it is possible that certain algorithms will provide better results for some metrics than for others. The results from these tests will also be examined, to identify any improvements that could be made with the algorithms, to increase their suitability for implementation on network processors.

Once this has been done, the study will research the hardware design of network processors. This analysis will help determine the suitability of network processors for providing packet classification. The study will compare network processors to traditional hardware architectures, which are being used for similar purposes. Once these differences have been noted, the study will be able to determine the suitability of a particular network processor with regards to packet classification. This part of the study should also led to recommendations for any improvements that could be made on future network processors to increase their suitability for the task of packet classification.

Lastly, the feasibility of the network processor implementation will be examined, to determine how suitable it would be for real-world implementation, particular attention will be paid to the implementation of a QoS-enabled router.

[1]These metrics will be further examined in chapter 2.5.

1.3 Scope and Limitations

Several limitations have been set to reduce the scope of the study:

This study will focus on the Intel IXP2400 network processor [23], this was based on the research group's collaboration with the Intel IXA University Program. Although the implementation will be done on only this processor, most network processors have a similar architecture and the information obtained should pertain to them as well. The specific evaluation board that will be used is the ENP-2611 from Radisys [44].

Packet classification is an issue that is applicable to a number of different fields in networking, however this paper will focus on the use of packet classification in providing QoS, namely the DiffServ architecture. This has been done to provide a framework in which the results can be evaluated – this being said, the issue of packet classification will be looked at in as general a sense as possible, and the results should be applicable to other applications.

The metrics used to evaluate the packet classification algorithms will be: classification speed and memory requirements, as these are the parameters that are the most applicable to the DiffServ architecture. The update complexity and preprocessing time metrics will only be discussed in minor detail.

The packet classification algorithms will be implemented entirely in software, while it may be possible to improve performance by implementing some, or all, of the algorithm as dedicated hardware-units. The study has been undertaken to show the performance of the algorithms on existing network processors. Information will be given regarding hardware improvements to future network processors, but said improvements will not be evaluated.

1.4 Thesis Outline

The remainder of this document is organised as follows:

Chapter 2 will provide the literature review that will lay the foundation for the rest of the study. The structure of the TCP/IP protocols will be examined in more detail and observations will be made regarding their ability to provide QoS differentiation. Once the limitations of the existing architecture have been identified, the study will provide information on the architectures that have been proposed to solve these problems. Particular attention will be given to the DiffServ architecture as this is the protocol that will form the

reference point for the study. Once an understanding of QoS has been established, some background information on network processors will be introduced. Current network processor architectures will be examined, and information regarding their specialised features will be presented. This chapter will look into the details regarding packet classification. This is an area of study with a large amount of scope, and there are a number of different algorithms that will be looked at. Algorithms will be divided into a number of different categories based on the methods they use to classify packets. Algorithms in these categories will then be looked at as a whole, because of their similar nature. The algorithms identified for implementation will be looked at in more detail as will as the algorithms that contain features worth noting.

Chapter 3 takes the algorithms mentioned in Chapter 2.5 and gives the details of their implementation on the network processor. This is an important process because though there are multiple ways of implementing an algorithm, one must be aware of the resource that the network processor provides in order to use these to their full potential. Several issues regarding the implementation or these algorithms are also be presented in this chapter.

An overview of the evaluation framework that has been used is given in Chapter 4. This includes details of the particular hardware and software, as well as details regarding the test data used when evaluating the packet classification algorithms. This test data includes various sized packet classification rule-sets, used to populate the databases of the packet classification algorithms, and generated trace packets, to test the reliability of the algorithms when receiving packets.

In Chapter 5 the results collected from the evaluation framework are discussed. These results will be interpreted and used to evaluate the implemented packet classification algorithms. This study will use the metrics specified in Chapter 2 to evaluate the performance of each algorithm and details will be given outlining the significance of the results.

A series of conclusions are presented in Chapter 6. These conclusions are drawn from the above mentioned results, and given to address the issues raised throughout this study. This chapter also lists any recommendations that were noted during the course of this study. These recommendations will include suggestions of future areas of study, which were outside the scope of this work. These recommendations will also be given to allow one to extend the results obtained in this study to future work.

Chapter 2

Literature Review

2.1 Introduction

The previous chapter sought to introduce some of the principles of IP networks and show how the need for QoS mechanisms has arisen over the last few years. This chapter intends to provide a more detailed discussion of these topics, addressing the fundamentals of the Internet Protocol and showing how QoS schemes have been introduced to provide service level differentiation over these networks. The last part of the chapter will give an overview of the network processor architecture, describing the core features that make them suitable for higher-level packet processing tasks.

2.2 IP and QoS

With the advent of services like VoIP, a great deal of attention has been paid to providing QoS over IP. Although most of these real-time services have only gained popularity fairly recently, the originators of IP foresaw the need for service differentiation when the protocol was created. It is doubtful that they envisioned the levels to which this would be taken, but nevertheless they provided a mechanism in which a particular packet's QoS requirements could be specified.

This mechanism was implemented by provisioning a field in the IPv4 packet header, in which the service-level requirements could be encoded. This 8-bit field was called the Type of Service (TOS) field and its position in the IP header is shown in Figure 2.1.

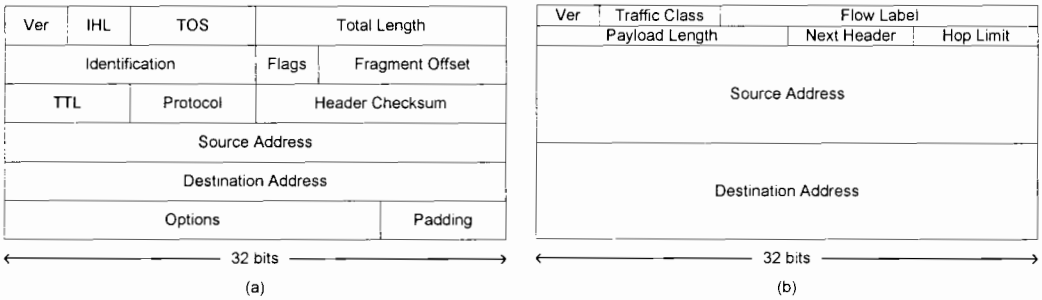

Figure 2.1: The IP packet header fields: (a) IPv4 headers, (b) IPv6 headers

The original format of the TOS field was provided along with the IP specification in RFC 791 [21]. However, it was later found that this specification did not provide enough clarification on certain aspects of the octet, and RFC 1349 [2] was released to address these issues.

Due to the fact that the network (IP) layer of a packet has no knowledge of the particular application that it is carrying, routing decisions are made purely on the information contained in the IP header. This can lead to problems because different network routes can have different QoS limitations and therefore the best route defined by the network layer may not be the best route for the application. This is possible when, for instance, a particular route has very high delays and the application is of a real-time nature and requires a connection with a low delay. The TOS field was therefore introduced to allow the application layer to communicate QoS needs directly to the network layer. Figure 2.2 shows the layout of the TOS field and how it is composed of three parts, these sub-fields are explained in more detail below:

- The first sub-field is known as the *Precedence* field, this specifies the importance of the traffic. This allows routers to place a higher priority on network control packets than on standard data packets and ensure that they are delivered, even when the network is under high-load. The IP specification assigned each precedence code to a particular type of traffic. Certain codes were used for intra-network control, others for inter-network control and still others for standard or routine traffic.

- The *Type of Service* (TOS) sub-field was defined as a 4-bit wide field, where a specific integers corresponded to particular parameters which could be prioritised in order to try and meet the QoS requirements of the application. An application could choose to

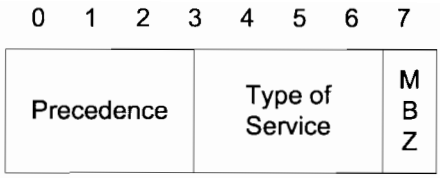

Figure 2.2: The Type of Service (TOS) field of the IP header

try and either minimise delay, maximise throughput, maximise reliability, minimise monetary cost or, the default, provide normal service. While the original specification of this field defined it as a bit-field in where more than one of these parameters could be set, the 1992 revision, redefines the field so that only one parameter can be prioritised at a time.

- The sub-field marked *Must-Be-Zero* (MBZ) is not used and should be set to zero. It was reserved for future expansion.

Although the TOS field was introduced to provide a mechanism for QoS, it does not provide absolute bounds with regards to the service level provided. In other words, even if a traffic stream uses the field to specify service requirements, the network makes no guarantee that these requirements will be met. The specification states that although a user requests, for example, the maximum throughput, the route that is chosen to service this request is based on limited information and might not be the absolute maximum. The specification states that the TOS field is intended to provide better service when available. It is therefore possible to receive service levels equal to those which would have been achieved otherwise, had the TOS field been left blank. This ensures that traffic is not rejected if the required service levels cannot be met.

Due to the vague specification of the TOS field in the original IP specification, the real-world implementation of this portion of the protocol is very limited. [2] was published to provide a better understanding of how implementation should be approached, but uptake was still minimal. A study performed in 2000, found that only 10 percent of traffic collected on the Internet contained a TOS value, different to the default [33]. Of this 10 percent, it was found that the majority of it was composed of only 4 discrete values. Diffserv redefines the use of the TOS field previously defined in the IP header.

2.3 Differentiated Services (Diffserv)

Unlike the original TOS field in the IP header, the Diffserv architecture seeks to provide absolute bounds on the service-levels of the network. The *Differentiated Services Working Group* was set up by the IETF to provide a system that can offer service differentiation in a relatively simple and coarse manner.

The *Integrated Services* architecture functions by using a signalling protocol, known as RSVP [7], to setup virtual links, which can provide the required QoS, between nodes. This is often impractical when dealing with short-lived flows, as the overhead required to setup a connection is often greater than the transmission of the flow. Another problem with this approach is that, when operating in the core of the network, there may be a great number of flows requiring QoS bounds. If each core node is required to keep state information for each micro-flow, the memory and processing requirements will become the bottleneck of the system, slowing the core network down considerably.

Diffserv seeks to alleviate these problems by aggregating traffic flows and providing QoS on a node-by-node basis. This is done by classifying traffic as it enter a Diffserv-enabled network based on the stream's QoS requirements. Traffic streams with similar requirements are grouped into what is known as a *Behaviour Aggregate* (BA). Each BA is then marked with a particular Diffserv Code-Point (DSCP), which can be examined by the nodes in the network to determine the level of service that the BA requires. This has the benefit by allowing a BA to be treated as a single flow, therefore reducing the overhead required in the core network nodes.

This approach does not, however, remove the overhead entirely. In aggregating the traffic, and therefore simplifying forwarding in the network core, the complexity is moved to the edge of the network. In order for traffic to enter a DS-enabled core network, where a certain level of QoS can be guaranteed, the traffic streams must firstly be classified to determine which level of service they require, the streams must then be monitored to ensure that they adhere to the traffic profile in which they are assigned. If this is not the case, a stream may need to be shaped, so that it does conforms to its profile, or dropped entirely.

Once a traffic stream has entered the core network, packets are dealt with on an individual basis. Each packet header is examined, to determine the BA assigned to the packet, and the network node then makes the necessary scheduling and queueing decisions based on the packet's BA. If the QoS-bounds for each packet are not violated at any of the network

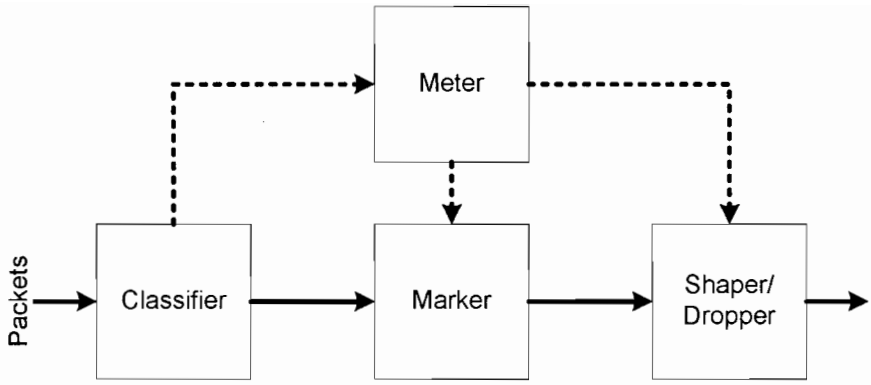

Figure 2.3: Functional structure of a Diffserv-enabled network node

nodes, then the overall end-to-end QoS requirements for the traffic stream will have been met.

2.3.1 Architecture

Diffserv (DS) provides a comprehensive service-differentiation architecture in that it defines not only the packet encoding, but the entire system. The architecture is based on a number of different elements that are implemented in the nodes of a network. These nodes perform different functions, based on their position in the network, and work together to provide an end-to-end QoS solution.

The Diffserv architecture specification [6] defines 4 elements that must be implemented in order to provide QoS bounds for applications. These elements are the *Classifier*, the *Meter*, the *Marker* and the *Shaper/Dropper*. Figure 3.1 shows how these blocks function together in a DS-enabled node, the solid line represents the flow of packets through a node and the dashed line represents state information being passed between elements.

The first step, once a packet has been received, is to classify the packets with regards to their QoS requirements. This is done by comparing the values in the received packet header, to those in the packet classification database of the node. There are two different levels of packet classification that take place on a DS domain, the first is known as Multi-Field (MF) classification, this happens at the edge of the network, and is used to determine the BA of a packet. MF-classification involves examining multiple fields in the incoming packet header, and is the area of packet classification that this research is focused on. Once the BA of a packet has been assigned, the interior network nodes will only need to perform

BA classification, where the BA label of the packet is used to identify the service level of the packet. This is a much simpler problem as the classification algorithm is dealing with one field rather than multiple fields as in the MF classification stage.

Information regarding the temporal properties of a traffic stream are collected by the Metering element. This information is used to ensure that a traffic stream remains true to the specified traffic profile. This element does not directly manipulate the packet stream, but it is responsible for providing information to the other elements of the node. RFC 3290 describes a leaky bucket architecture that can be used to measure the temporal properties of a stream and then provide information to the Shaping and Marking portions of the DS-enabled node [5]. There has been a fair deal of research into this area and several algorithms and architectures have been proposed to provide this functionality [45, 18, 19].

Once the Marker has determined which BA a particular packet should be assigned to, the packet must be marked so that this information can be passed on to other nodes in the network. This is done by marking the packet with a particular *DS-Code-Point* (DSCP) [39]. This code-point corresponds to a *Per-Hop Behaviour* (PHB) that must be observed at all nodes in the network, in order to ensure end-to-end QoS requirements being met. The marker will take into consideration information passed to it from both the Classifying and Metering functional blocks. If the Metering unit identifies that a particular BA is out of profile, the marker may choose to mark the packet with a different DSCP than it would have normally. If a packet is in profile, the Marker will usually mark the packet based on the identification made by the Classifier.

The Shaper/Dropper functional block is responsible for meeting the QoS requirements that a stream has been assigned to. This functional unit will implement the buffer management and packet scheduling portions of the node. These are extremely important tasks as they will ensure that the traffic stream remains in profile and is able to meet their service level requirements. Once a packet has been classified and marked, it is placed in a queue before being transmitted to the network. The shaper is responsible for managing these queues and determining the order in which packets are transmitted. A traffic stream that has higher QoS requirements, will have its queues serviced more often than a stream with lower requirements. A great deal of research has gone into buffer management [36, 34, 12] and packet scheduling [51, 32, 27, 8, 29, 46] algorithms, as these issues have been around for quite sometime in ATM and other networks. Scheduling in IP poses a slightly different problem in that, unlike ATM, IP allows the packet size to be varied, the implications of these are discussed in [51] and other papers.

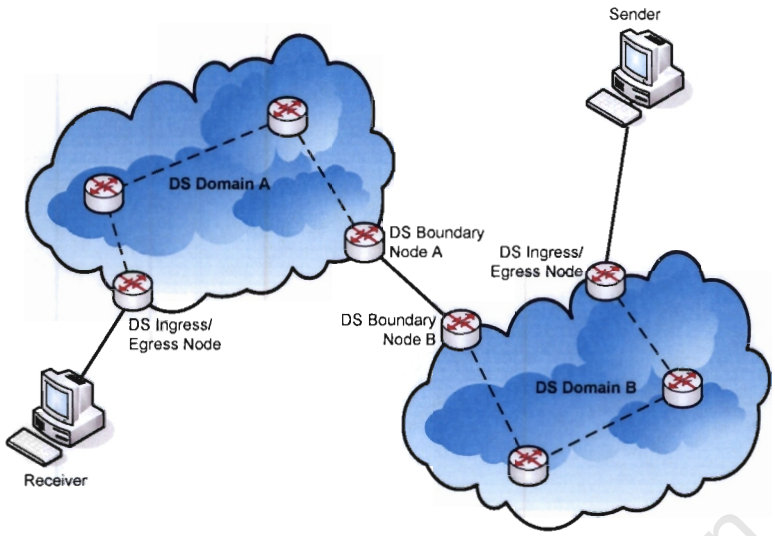

Figure 2.4: A typical DS-enabled WAN

Although all of the functional blocks mentioned earlier are required in order to provide DS-enabled services, these need not be implemented, to the same degree, on every node of the network. Figure 2.4 shows the layout of a typical DS-enabled network, this network consists of two smaller DS-enabled networks that are connected together over a single link. Each DS-enabled network, or DS-domain, contains a number of DS-enabled nodes, these nodes are known as DS interior nodes as they operate on the interior of the DS-domain. The DS boundary nodes A and B connect the two domains together, while the ingress and egress nodes connect the sending and receiving hosts to the DS-enabled core-network. The dashed lines represent the routing path taken by traffic travelling between the two nodes.

In Figure 2.4 we are presented with two hosts which wish to communicate using some application with QoS requirements. Since both hosts are on different DS-domains, not only must each domain provide a constant level of service, but the network should provide bounds on the end-to-end QoS of the connection.

As traffic enters a domain through the DS Ingress Node, the traffic stream must be conditioned to ensure that it fits the traffic profile to which is belongs. The ingress node is responsible for identifying the service-level requirements of the traffic entering the domain by performing some form of multi-field packet classification. This classifier may use application layer specific header fields, or more likely network and transport layer headers. This will allow traffic to be marked and the temporal properties of the stream examined and conditioned – this may include shaping or dropping packets in the stream. The ingress

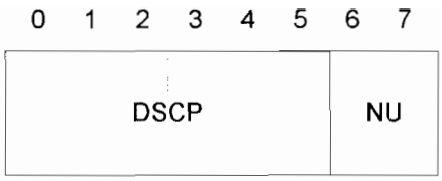

Figure 2.5: The DS-field in the IP header

node must then be marked with a DSCP, so that they can be identified in the rest of the network .

The marked packets can now be treated in an aggregated manner by the rest of the network, allowing the BA of a particular packet to be identified by examining the DSCP in the packet header. This allows the interior nodes to retain their simplicity and focus on the task of scheduling and forwarding packets. The DS-domain should operate on the same traffic scheduling parameters ensuring that a stream receives the same level of service, regardless of the routing path through the network.

Since it is possible for bordering DS-domains to use different DSCPs and provide different levels of service, it becomes necessary for BAs to be translated when they pass between domains. Neighbouring domains should have a Service Level Agreement (SLA) that contractly states the service levels that each network guarantees to traffic originating from a neighbouring domain. The SLA should also contain information regarding how traffic entering the network will be conditioned, this can include changes made to the DSCP and temporal restrictions on the aggregate stream.

The DS egress node will be the final point of the DS-domain that a packet translates. Once a packet leaves a DS-domain, the DSCP of the packet can be ignored, or removed, and the packet is forwarded to the receiving host.

2.3.2 Per-Hop Behaviours

Diffserv identifies traffic streams by marking packets with what is known as a Diffserv-Code-Point (DSCP). The DSCP is placed in the DS-field portion of the IP header; this field has replaced the TOS field in the IPv4 header and the Traffic Class field in the IPv6 header. The layout of this field can be seen in Figure 2.5.

Each DSCP maps to a single PHB, while this PHB might not be unique to a particular DSCP, all DSCPs that map to a PHB will receive the same level of service. The PHBs

are stored in each node on a DS-domain, and contains information regarding the QoS requirements of the particular traffic classes. These requirements will denote delay bounds, bandwidth requirements, etc. A DSCP and corresponding PHB need only be unique to a particular DS-domain as the DSCP can be re-mapped when traffic is transmitted between domains. Although this is the case, there are several DSCPs that have been standardised. These are the *Expedited Forwarding* (EF) class [24] and the *Assured Forwarding* (AF) class [17].

The EF class is intended to function as a virtual leased-line class. EF provides QoS bounds similar to those that would be seen on a leased line. These include fixed bandwidth availability, a small delay and minimal jitter. Since delay and jitter both result from queueing delays, packets of the EF class should be treated to no, or little, queueing. This class is available for premium services like VPNs, where a company wishes to replace a physical leased-line with an equivalent service. EF implements a strict traffic conditioning profile, where any traffic exceeding the specified levels is dropped.

The AF class provides a mechanism for networks to offer service levels greater than that of best-effort traffic. The specification defines four different service classes, where each class is assured better service than the class below it. Within these classes, there are three drop-precedences defined – this gives a total of twelve DSCPs. The AF specification suggests assigning a higher drop-precedence to packets once the stream exceeds the limits specified in its traffic profile. These packets are then given less priority than other packets of the same class, and can be dropped first in case of network congestion.

Although the AF and EF classes are the only two PHBs defined, they provide a great deal of flexibility on a DS-enabled network. This flexibility is also enabled due to the existence of DS-domains. The same PHBs could provide different QoS bounds on a number of different networks. However, since PHBs are re-mapped at DS ingress and egress nodes, this can be taken into account and a PHB can be changed when it enters a new DS-domain. This allows a packet to experience the same level of service on both networks, as well as enabling the reuse of DSCPs. This is especially useful due to the fact that the DS-field is only 6-bits wide and therefore only provides space for 64 unique DSCPs.

To move from an architecture specification (like the Diffserv RFCs) to a workable implementation, the correct hardware needs to be used for the task.

2.4 Network Processors

Network processors (NPs) have provided an architecture that can handle the increased packet processing and line speed requirements of DS-enabled networks. This is done, while still maintaining the programmability of a software solution. These features give NPs a clear edge over ASICs and other hardware-level solutions, which have traditionally been used in the high-speed realm of core and edge networks.

ASICs have earned a place in the network processing market due to their ability to process packets at extremely high speeds. This is possible because of the fixed-hardware nature of these solutions. This speed does come at the price of a great deal of flexibility; since ASICs are designed as hard-wired, integrated devices, it is very difficult to change the unit once it has been designed. Introducing new functionality or updating existing functionality requires the hardware design of the ASIC to be changed. This leads to very high development times, as changes cannot be readily made, and once they have been made, the unit must be re-manufactured before the extra functionality can be added to a product. It is not uncommon for ASICs to require development cycles of longer than 18 months [20]. This makes them largely inadequate for tasks like Diffserv and packet classification, where there is a great deal of research happening and architecture changes being made.

FPGAs provide more flexibility than ASICs, in that it is possible to re-program them. This allows for shortened development times, though still not quite as short as software-based designs. There are a few disadvantages of FPGAs including: high power consumptions, high cost and fixed architecture. FPGAs have much lower packet though-puts than ASICs, this makes them relatively unsuitable for high-speed network applications.

On the other extreme of the flexibility/speed tradeoff, there is the General Purpose Processor (GPP). GPPs provide a great deal of flexibility in that they are software-programmable devices, however, this comes at the cost of speed. Network packet processing requires packets to be processed at high line-speeds, which GPPs can not achieve. Although, they will be sufficient for lower speed networks, their design does not include a great deal of scalability.

Network processors have been introduced to provide a balance between these two extremes. These devices are software-programmable, allowing them to offer a great deal of flexibility, and have a hardware architecture that is specifically designed to process packets at line-speeds. This enables them to have rapid development times, while not sacrificing any of the speed needed by today's high-speed networks. Shah compares these four technologies

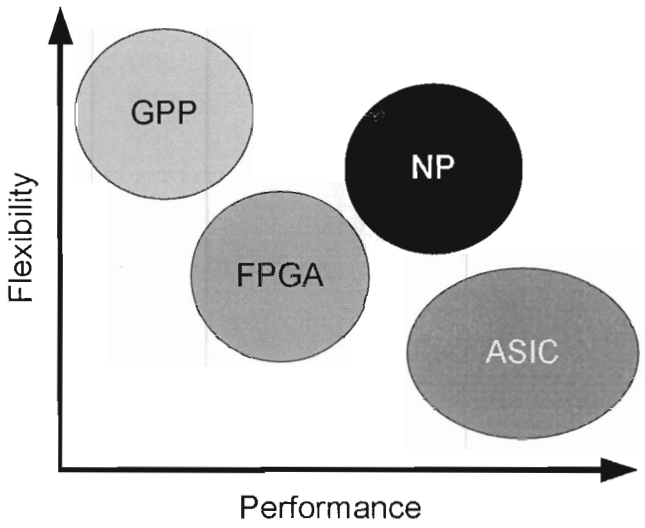

Figure 2.6: A comparison of different technologies, showing the trade-off between flexibility and performance

with regards to their performance and flexibility in his Master's thesis [49] and the results are shown in Figure 2.6.

Although there are a number of different NPs currently on the market, they all employ similar mechanisms. These are discussed below, in order to give an understanding of how NPs differ from traditional network processing architectures.

2.4.1 Hardware Mechanisms

NPs have a number of different mechanisms that enable them to process packets at line speeds. The implementations of these mechanisms differs from processor to processor, but all rely on the same basic principles. This section will discuss some of these processing mechanisms and give details about how they are implemented in the IXP2400 [22].

Packet-switched networks offer a high level of data *parallelism* due to the fact that packets can be considered on an individual basis. There is little, to no, data dependency between sequential packets as each packet contains a header with all the information needed to process it[1]. This allows the task of processing the packets to be performed in parallel and network processors have hardware to take advantage of this. This is usually done by

[1]This is not always true as packet processing done at the application layer may introduce inter-packet dependencies. This is also the case when packet fragmentation has taken place.

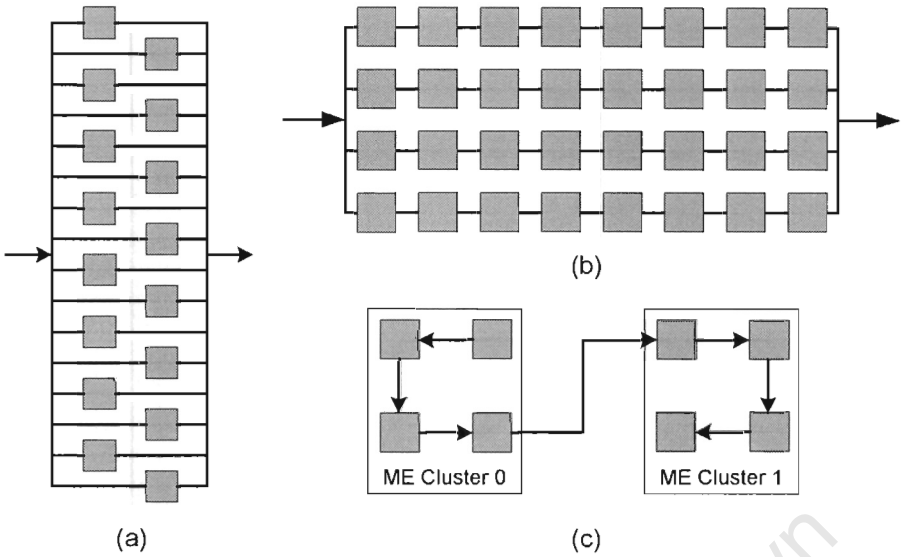

Figure 2.7: The layout of PPEs in different NPs. (a) Lexra NetVortex, (b) Cisco PXF and (c) Intel IXP2400

means of multiple Programmable Processing Engines (PPEs). These engines are essentially RISC processors, with added instructions, that execute in parallel. PPEs are the main data-processing unit of the NP and although they are present in most NPs, different NP architectures have these engines in different numbers and layouts. Architectures like the *Lexra NetVortex [13]* have up to 16 MIPS cores that operate in parallel, while the *Cisco PXF [62]* arranges its 32 processors in 4 pipelines. The NetVortex will assign a single packet to each PPE as it becomes available, the PPE then processes the packet to completion. In contrast to this, the PXF will assign packets to its four pipelines, and packets are passed between different PPEs until they reach the end of the pipe.

The IXP2400 provides a balance between these two layouts; it provides 8 PPEs (known as microengines) that can either be arranged in a pipeline or parallel manner. The microengines implement what are known as next-neighbour registers; these allow data to be passed to the next microengine in the chain to allow pipelined processing. Figure 2.7 contrasts these three layouts and shows how each is implemented.

NPs also provide another level of parallelism by implementing multi-threading on the PPEs. Due to the vast difference between the clock cycle time of the PPEs and the access time of on-board memory, the PPEs will often have to wait for multiple clock cycles before data is returned from (or written to) memory. NP manufacturers have added functionality in

order to allow PPEs to execute multiple threads concurrently. The *Agrere PayloadPlus* [61] is the NP that currently supports the largest number of threads with 64 contexts per PPE. The IXP2400 supports fewer contexts, with a total of 8 threads per microengine (ME). Many NPs also implement no-overhead context switching to allow active threads to be swapped into, and out of, execution with no clock-cycle penalties. The IXP2400 provides this functionality.

Secondly, the *memory architecture* of NPs is somewhat more complex than that of GPPs. This is due to the unique architecture of NPs and their highly parallelised design. NPs have a number of different memory requirements, including memory for storing program code, packet data, queueing information and other algorithmic data-structures. These different types of data each have unique requirements. For example, program code needs only a small amount of memory but needs to be accessed at very high speeds while packet data requires much larger storage sizes, but at lower throughputs. Typically, a NP will have both internal and external memories. The internal memory will satisfy the small, fast memory needs, while externally there will be space for larger, albeit slower, memory. The IXP has four different types of memory: local memory, scratchpad memory, SRAM and DRAM.

NPs will generally contain a number of *co-processors* that operate alongside the PPEs. These hardware-level units are designed to provide a greater level of performance for common tasks like queue management, CRC calculation and encryption. These co-processors are usually on the same die as the rest of the network processor, connecting to the main bus. This allows them to have high throughputs and easy access from the PPEs. Although this hardware can speedup common tasks dramatically, they provide no additional performance if the specific functions that they implement are not used by the NP application developer. The IXP2400 provides additional hardware for hashing, linked-lists, ring operations and CRC calculations. It also provides a sixteen-entry CAM unit per ME.

A mechanism that does not specifically enable high-performance packet processing, but forms part of the basic functionality of NPs includes the *Media and Switching Fabric interfaces* (MSF). These enable the NP to receive packets from the network and route them to other networks. NPs typically contain a configurable network media interface that allows them to connect using a number of different MAC protocols.

All the components of the NP are connected to a shared bus, allowing communication between the different functional blocks. Figure 2.8 shows the architecture of the IXP2400.

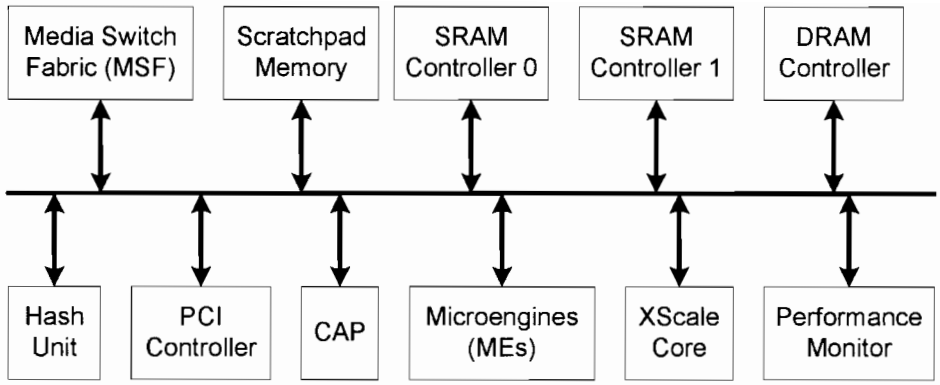

Figure 2.8: A functional overview of the IXP2400 network processor

2.4.2 Instruction Set

In order to provide line-speed packet processing, NPs implement a number of instructions that are specifically designed for packet processing. These instructions will be part of the instruction set of the PPEs and provide an efficient means of executing commonly used tasks.

There are generally two types of instructions present in PPEs: internal and external instructions. Internal instructions are instructions that are implemented in the silicon of the PPEs. They include the common instructions for using the ALU of the engine, accessing memory, and receiving and transmitting packets. It is also possible to have more exotic internal instructions, for example, the IXP2400 contains an instruction that will return the number of the first bit set in a register. This kind of operation would usually take a number of instructions and branches to implement, yet the IXP2400 implementation is able to return the correct value in one cycle. External instructions provide a mechanism for accessing the different units of the network processor, the IXP2400 contains instructions that allow the PPEs to forward data to, and receive data from, the hashing, memory, MSF, etc. units of the NP. Inter-unit communication is performed using hardware-level signalling that allows a unit to signal the PPE when a function has finished executing.

2.5 Packet Classification

While longest matching prefix lookup on IP addresses has been around in routers for quite some time, the idea of multi-field packet classification is relatively new. It has therefore

Rule No.	Source IP Address	Destination IP Address	Destination Port	Transport-layer Protocol	Action
1	134.45.34.*	155.12.67.4	80	TCP	Permit
2	134.45.34.*	155.12.67.*	1 - 1024	TCP	Deny
3	134.45.65.22	155.12.*.*	21	TCP	Permit
4	*	155.12.128.7	1 - 65535	UDP	DSCP 5

Table 2.1: An example 4-dimensional packet classification rule-set.

been an area into which there has been a great deal of research [31, 4, 16, 52, 3, 59] in the last few years.

Multi-field packet classification – or just packet classification, as it is commonly called – is the task of examining the fields in a network packet, to determine if the packet conforms to a particular rule in a rule database. The fields that a classifier examines can be varied, though generally they will be the classic IPv4 five-tuple fields. These are: the source and destination IP address, the source and destination transport-layer port numbers and the protocol type. These fields, present in the headers of a packet, can be used to identify a majority of IP traffic. This being said, most packet classification algorithms are not limited to this five-tuple and can use a number of fields for classification, including application-layer fields.

Traditionally, the main use of packet classification, was in edge-network firewalls. These firewalls would be responsible for monitoring the traffic that enters and leaves a network and dropping unwanted packets. The rule databases in the firewalls studied have generally been fairly small – the largest database encountered by [15] had about 1700 rules – however, with the advent of QoS architectures like Diffserv, this number is expected to increase. As noted in chapter 2, packet classification plays an important part in identifying traffic streams in order to provide differentiated levels of services. Diffserv-enabled edge routers are required to classify all traffic, not just malicious traffic; this will lead to much larger rule databases than are currently seen. This will be especially true of classifiers that use more than just the network and transport-layer fields of a packet for identification. While it is impossible to know for certain the size of these future databases, some sources [4, 10] envisage them to be in the tens to hundreds of thousands; it is therefore of interest to provide solutions that cater for these large rule-sets.

An example packet classification database is shown in Table 2.1. This database is for a four-dimensional classifier that could be found in a firewall or DS-edge router. The Action

Rule	x Range	y Range
R1	0 - 31	0 - 255
R2	0 - 255	128 - 131
R3	64 - 71	128 - 255
R4	67 - 67	0 - 127
R5	64 - 71	0 - 15
R6	128 - 191	4 - 131
R7	192 - 192	0 - 255

Table 2.2: An example 2-dimensional packet classification rule-set.

column lists the desired action to be taken when a packet matches the particular rule, these actions specify whether to transmit, drop or re-mark the DSCP of a packet. It is common to see classifiers with overlapping rules – in the example database Rule 1 is a subset of Rule 2 – it is therefore necessary to give a priority precedence to overlapping rules. In most cases this is done by assigning priorities in descending order with Rule 1 having the highest priority and Rule N having the lowest priority. It can also be seen that some fields of the database are specified as prefixes (e.g. the IP addresses), while others are specified as ranges (e.g. the transport-layer ports). Range matching is a more complex procedure in that it requires almost three times as many transistors-per-bit as longest prefix matching [64].

Packet classification is an inherently difficult problem to solve and is equivalent to the identification of an enclosing region of a point in a k-dimensional space. This can be seen by looking at the 2-dimensional classifier database shown in Table 2.2. Each rule in the database can be mapped to a 2-dimensional region as shown in Figure 2.9 and an incoming packet will map to a single point in the space. It is therefore possible to get performance bounds for the problem of packet classification from the area of computational geometry [41]. It has been shown that the problem is bound by either $O\left(\log^{k-1} N\right)$ time complexity and linear storage space, or $\log N$ time and $O\left(N^k\right)$ space; where N is the number of regions (or rules). To put this into real terms, 1000 five-dimensional rules would require in one extreme, 10K memory accesses per packet and in the other extreme, 1000 GBs of storage space; both of these are impractical for current routers.

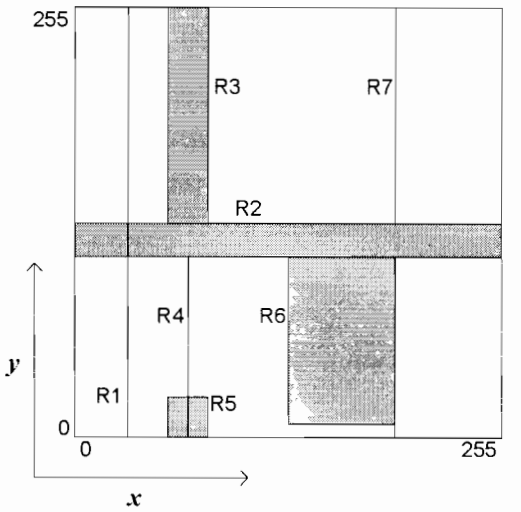

Figure 2.9: The mapping of an example rule-set onto the 2-dimensional region.

2.5.1 Performance Metrics

There are a number of different metrics that can be used to evaluate packet classification algorithms [16]:

- Arguably, the most important metric in determining the effectiveness of a packet classification algorithm is the *search-speed*. In order to provide differentiated services at line-speeds, the time complexity of algorithms needs to be much less than the $O\left(\log^{k-1} N\right)$ maximum bound specified earlier. Due to the current disparity between processing and memory speeds, the time complexity of algorithms is dominated by the number of memory accesses required per classification. It therefore stands to reason that algorithms which require fewer memory accesses will have better search-speeds.

- The other side of the problem is the *storage requirements* of the algorithm. According to [41], in order to increase the search-speed of an algorithm, there is a tradeoff with regard to storage space. Obviously, routers do not contain infinite amounts of memory and therefore algorithms must have reasonable storage requirements. The memory technology to be used is chosen with respect to three parameters: cost, speed and size. Current technologies can only ensure two of these parameters at a time – e.g. it is possible to produce memory that is both fast and large, but it will be expensive. Therefore because of the stringent time bounds for packet classification,

algorithms need to have relatively low storage requirements in order to allow cost-effective memory to be used.

- *Updatability* refers to the ability of a packet classification algorithm to allow rules to be added to (and removed from) the classification database. This ability depends on the data structures employed by the algorithms. Certain data structure allow fast updates, while others require the data structure to be rebuilt from scratch every time the rules are changed. This parameter is less important than the other two, as some applications have relatively static classification databases and do not require a high rate of updatability. This is true for Diffserv due to the aggregated manner of the architecture, new rules do not need to be recreated for each micro-flow however, the architecture does provide space for this [5].

This study focuses on two of these three metrics, namely: search-speed and storage requirements. Updatability is ignored to a large part, due to the assumption that the packet classification databases of the DS-enabled nodes will be relatively static.

2.5.2 Heuristics

Part of the complexity of packet classification, as shown in Section 2.5.1, can be negated by exploiting the structure of the packet classification databases found in real networks. Various studies [4, 14, 15, 59, 3] have shown that there is an inherent structure in packet classification databases – this was determined by statistically analysing databases obtained from various company firewalls. It is therefore possible to use a heuristics-based approach that allows packet classification algorithms to have a time complexity that is much better than the worst case complexity found in literature [41]. It is important to note that although a heuristics-based algorithm provides better performance than traditional algorithms, this performance is heavily linked to the structure of the classification database being used. If a database does not conform to the preconceived structure, which the algorithm is trying to exploit, the performance of the algorithm will be greatly reduced.

2.5.3 Algorithms

There have been numerous algorithms proposed to meet the high demands of packet classification. These algorithms differ greatly in regard to the mechanisms that they use for

classification. Taylor introduces a taxonomy to classify these algorithms according to their high-level approach to the problem [64]. The four approaches are: the exhaustive search, decomposition, the decision tree and tuple space.

An *exhaustive search* is the simplest approach for packet classification and involves matching every rule in the database against the incoming packet. Although a large number of algorithms use this approach to a certain degree, there are very few that use it explicitly. The two most common ones that do are the linear search and TCAM.

Linear Search

In the linear search, each rule is stored once in the database. The algorithm then performs a linear search through the table starting at the rule with the highest priority moving towards the lowest priority rule. Once a rule is found that matches the incoming packet, the search is halted and the matched rule is returned. This algorithm is the most intuitive classification algorithm, however it only performs well for very small databases. This algorithm has a storage requirement of $O(N)$ as each rule is stored only once in the database. It also has a worst-case time complexity of $O(N)$ memory accesses.

Ternary Content Addressable Memory (TCAM)

TCAMs are hardware units that are capable of comparing a packet with every rule in the database in parallel. They differ from traditional CAMs in that they have the ability to store a "don't care" state, this is done by storing a bit mask alongside the regular data. This makes them particularly well suited for packet classification as this is the format in which rules are specified[2]. While TCAMs provide a mechanism for constant time lookups, they are inefficient in a number of other areas: the two main ones being power consumption and cost. Both of these are related to the extra logic needed to match each bit of a TCAM. Firstly, the extra logic leads to an increase in the amount of power drawn by the unit, this is in the order of 150 times more than SRAM [64]. Additional logic also means additional transistors and therefore additional die space. This leads to an increased cost to manufacture these devices.

Recently, Extended TCAMs (E-TCAMs) have been introduced to address the issues of power consumption and storage inefficiency. This is done by reducing the number of active

[2]Fields that are specified as ranges (e.g. port fields) need to be converted into bit-prefixes first.

areas in the device during matching and adding hardware to perform range matching [54]. This technology, though still relatively new, is a promising solution for hardware-based packet classification. However it will not be considered in this research as currently there are no NPs that contain TCAM units.

Decomposition arises from the observation that it is often more efficient to perform a number of single-field classifications in parallel and then combine the results at a later stage than it is to perform a single multi-field classification. During the single-field classifications, it is possible to have a single field of a packet matching a single field of a rule. Packets are said to match a rule only if all fields of the packet match all fields of the rule. It is therefore necessary to combine these individual field matches to determine which rules match every field of the packet. Once this has been determined the highest priority rule can be chosen. As long as the time complexity of performing the recombination is less than the time gained by parallelising the classification, there will be a beneficial net effect on search speed. This technique differs from the exhaustive search, in that rules are rarely stored in their original form. In order to decrease the processing time of combining the parallel single-field classifications, rules are usually stored in a pre-processed state. The format of this differs from algorithm to algorithm.

Parallel Bit-Vectors (BV)

The BV algorithm [31] seeks to split up each dimension into a number of intervals. An interval is defined by a range[3] on the dimensional axis in which the set of rules do not change. Therefore interval boundaries will only happen at the beginning and end points of a rule. A bit-vector, which lists the rules that overlap the interval, is then defined for the region. Each bit in the bit-vector represents one rule in the database, if the bit is set then that particular rule overlaps the current interval. An example of this mapping of regions and bit-vectors is shown graphically for a 2-dimensional database in Figure 2.10.

When packets arrive to be classified, the first step is to determine which interval each field of the packet belongs to. The most efficient means of doing this is by using a binary search algorithm, which requires $\log(2N+1)+1$ comparisons. Once this has been done for all packet fields, the bit-vectors for each field can be retrieved and combined to determine which rules match every field in the packet. The rule with the highest priority is then used

[3]In this algorithm, the word 'range' is used when referring to bit-prefixes as well as traditional ranged fields.

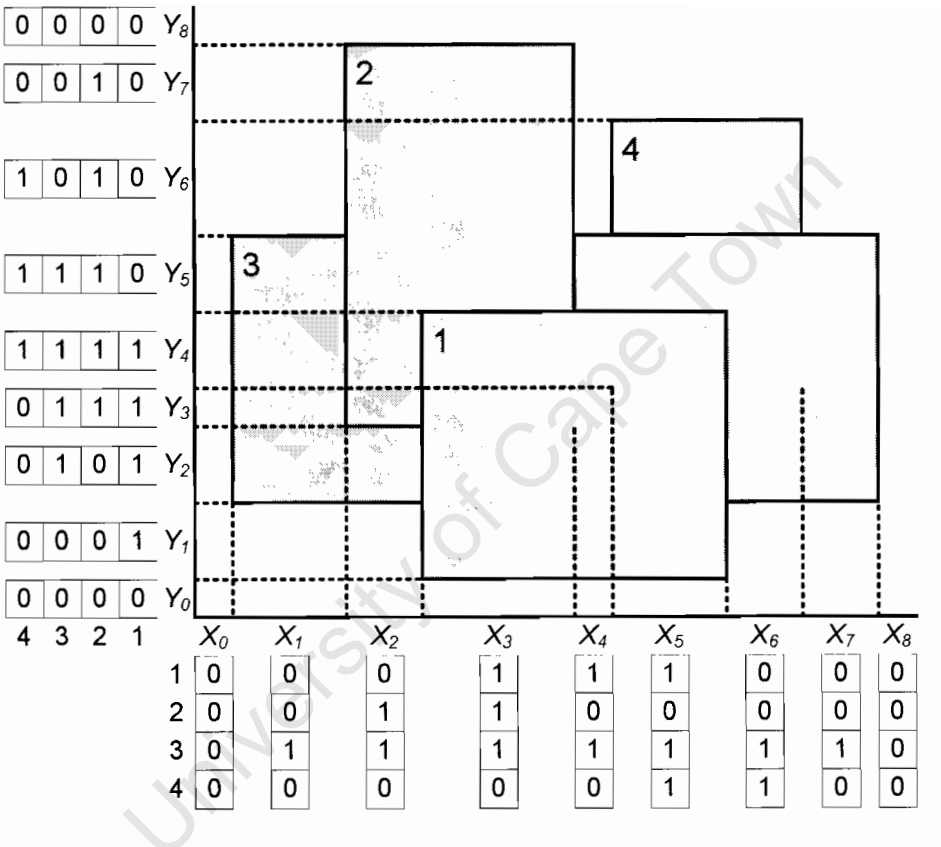

Figure 2.10: Example regions and bit-vectors for the BV algorithm .

to classify the packet.

The efficiency of the BV algorithm is due to the fact that each field in the classification can be performed in parallel and then combined at a later stage. This also leads to an efficient hardware implementation, in that the data structures for each field can be stored in different memory units and accessed in parallel. The storage requirements for this algorithm are quite large, needing a maximum of $(2N + 1)\,N$ bits per dimension. Lakshman and Stiliadis implemented the algorithm using an FPGA and five 128 KB SRAMs. This configuration was able to classify one million packets per second in the worst case, though the number of rules was limited to 512 [31].

Although the literature states that there is a maximum of $(2N + 1)$ intervals per dimension, the authors found that real classification databases contained far fewer than this. It was also found that the resulting bit-vectors were relatively sparsely populated. Several improvements to the algorithm were therefore introduced, both by the authors themselves and others [4]. These provided better average performance for the algorithms – with regards to both speed and storage.

Recursive Flow Classification (RFC)

Gupta and McKeown introduced RFC [15] as an algorithm that could map the string formed by concatenating the fields of an IPv4 packet header to a smaller string that represented the matching rule number of that packet. While this mapping could be done in one step, it is more efficient to split it into a number of phases and execute these phases recursively until a result is obtained. Initially, the packet header is broken up into a number of smaller pieces, or chunks. The size and number of these chunks is dependant on the structure of the rule database. These are then used as indexes into multiple tables in parallel memory units. Once results have been obtained from these lookups, they are used as indexes in later stages of the algorithm until a matching rule is obtained.

The main intelligence of the algorithm is in pre-computing the lookup tables and index chunks. Once this has been done, it is a relatively simple process to lookup the index for the next stage. It is also possible to combine several results from parallel lookups, and use this as an index. The overall functionality of the algorithm is illustrated in Figure 2.11.

While RFC provides fast packet classification, it has large storage requirements and preprocessing time for databases larger than 6000 rules [15]. The authors later proposed the

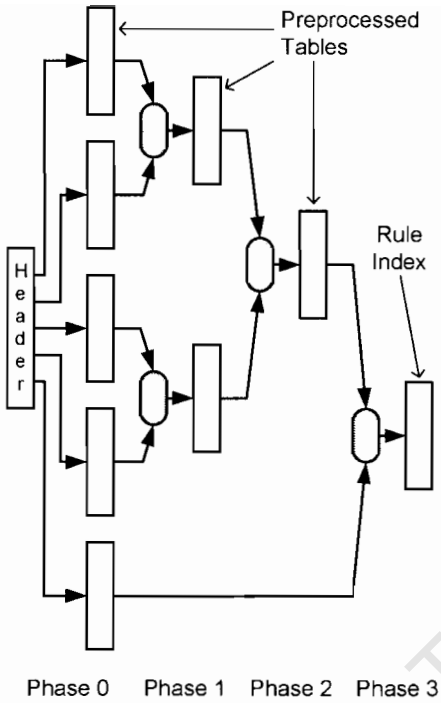

Figure 2.11: An example RFC layout using 4 phases.

HiCuts algorithm [14] as an alternative algorithm with lower storage requirements, though research has been done into compressing the RFC data-structures as well [55].

One of the more popular approaches to packet classification is the *decision tree*. In a decision tree, rules are distributed amongst the leaf nodes and the branches of the tree are traversed until one of these leaf nodes is reached. The fields of a packet are used as search keys when making decisions at each branch of the tree.

Hierarchical Intelligent Cuttings (HiCuts)

Hierarchical Intelligent Cuttings (HiCuts) [14] is one of the leading algorithms of this class. It uses a heuristics-based approach, where the search space is partitioned in a geometrical manner in order to create a decision tree. The decision tree is then navigated until a leaf node is reached, there a list of rules that overlap the particular geometric region of the search space are searched linearly until a rule is matched.

A key feature of this algorithm is the pre-processing step that creates the decision tree. The algorithm exploits the fact that real classification databases exhibit a great deal of "rule

clustering" – rule-sets will contain a number of rules which share similar field ranges. The idea is therefore to reduce the search area of the algorithm, in order to reduce the number of rules that need to be matched against a packet. There are two parameters that are used to determine the shape and depth of the tree. The first parameter, the rule threshold (*binth*) specifies the maximum number of rules that can appear in a leaf node while the space measure factor (*spfac*) is a measure of the amount of storage memory required by the node. Both *binth* and *spfac* are used to determine how many child nodes a node must be split into and in which dimension the cut should be made. The HiCuts algorithm can use a number of different metrics to determine the optimal dimension in which to make a cut, however research has shown that some of the metrics are more suitable than others [48]. Once the cut-dimension has been decided, the algorithm will split the node. The more cuts that are made at a node, the less rules in each child node and therefore the smaller the depth of the tree. However this also leads to the node needing more storage space as it more likely that rules will be duplicated in adjacent nodes. The *spfac* is used by the space measure function (*smf*) to balance these two factors.

An overview of the preprocessing step is given below:

1. Starting at the root node of the tree, if the node contains more than *binth* rules, cut the node. Use the *smf* and other metrics to determine how many cuts to make and in which dimension to make them.

2. For each child node, perform the same step recursively until the number of rules in the node is less than *binth*.

3. Once the rule-set has been partitioned, further post-processing can be performed to remove any redundant nodes and rules from the tree[4].

The following example is presented to give a better understanding of the HiCuts algorithm. The example 2-dimensional rule-set in Table 2.2 is shown graphically in Figure 2.9. For $binth = 2$ and $spfac = 2$ the rule-set can be mapped to the tree shown in Figure 2.12. The first parameter at the root node specifies the size of the region, the second identifies the dimension in which the cut has been made and the third shows how many parts the node has been split into. Once the root node has been split, since the second child of the root node contains more than *binth* rules the node is split again, this time along the y-axis.

[4]This step is not vital to the algorithm, but provides a way of reducing the storage requirements for the tree.

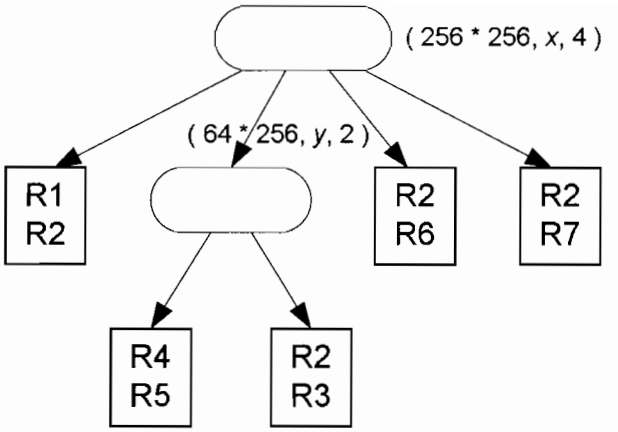

Figure 2.12: The HiCuts tree created from the rule-set defined in Table 2.2 using $binth = 2$ and $spfac = 2$

The heuristics used by HiCuts allow this algorithm to have the best balance between average tree depth (an indication of the number of memory accesses required by the algorithm) and average memory consumption of all the decision tree algorithms studied in [64]. A notable exception is the HyperCuts [52] algorithm that extends the HiCuts algorithm by allowing multiple cuts at each node. This allowed for reduced storage requirements for larger databases at the penalty of slightly more memory accesses.

The last category of packet classification algorithms is the *tuple space* algorithms. Srinivasan, Suri and Varghese published a paper [59] that introduced the notion of tuples. A tuple is said to be a collection of rules that share the same number of specified bits in each field. It was noted that while databases contained a large number of distinct rules, the number of distinct tuples was much less.

Tuple Space Search

The algorithm represented in [59] is the simplest method of searching the tuple space. Since the number or unique tuples is often much smaller than the number of unique rules, an exhaustive search of the tuples can be performed with fewer memory accesses than an exhaustive search of the rule-set. The preprocessing step of the algorithm involves first identifying the tuple associated with each rule and then placing each rule in the hash table of the tuple to which it has been matched. It is therefore possible to match a rule in M hashed memory accesses, where M is the number of unique tuples. Each tuple is searched

Rule	Specification	Tuple
R1	(00*, 00*)	(2, 2)
R2	(0**, 01*)	(1, 2)
R3	(1**, 0**)	(1, 1)
R4	(00*, 0**)	(2, 1)
R5	(0**, 1**)	(1, 1)
R6	(***, 1**)	(0, 1)

(a)

Tuple	Hash Table Entries
(0, 1)	{R6}
(1, 1)	{R3, R5}
(1, 2)	{R2}
(2, 1)	{R4}
(2, 2)	{R1}

(b)

Table 2.3: (a) An example rule-set and (b) its resulting tuple space hash table.

independently, and possibly in parallel, using a hashing algorithm to find the matching rule. The key used in the hashing function will be the same series of bits specified by the current tuple. Therefore the key extracted from the packet will be the same key that was used to hash a matching rule. The hash table stores each rules exactly once, this provides a storage complexity of $O(N)$. An example rule-set is given in Table 2.3a and the resulting tuple hashing table is given in Table 2.3b. There have been several proposed improvements to the tuple space search algorithm, namely the Pruned Tuple Space Search [59] and Entry Pruned Tuple Space Search [58]. Both of these algorithms seek to reduce the number of tuples that need to be probed, therefore increasing the search-speed of the algorithm.

The Pruned Tuple Space Search algorithm identifies applicable tuples by using a similar mechanism to the BV algorithm. The difference is that instead of a vector of rules being associated with each range, the bits of the vector now represent tuples. This reduces the number of tuples that need to be examined to determine a rule match.

Caching

While caching is not a packet classification technique in itself, it can be used with a number of existing algorithms in order to improve their average performance. Since this technique can only improve average performance, it is not very popular for use with QoS schemes, which require line-speed processing. This is because in order to ensure QoS bounds are not violated, packets need to be classified before they are queued. If a packet is placed in a queue before it has been classified, the queue management algorithm will not be able to give a packet with tight QoS bounds higher priority than any other packet, as all packets are still being treated equally. If worst-case classification of packets is happening at speeds

of less than line-speed, it is possible that QoS deadlines will be violated while a packet is still being queued for processing.

Caching is also becoming less effective as bandwidth increases. This is because a higher bandwidth results in more micro-flows and therefore less temporal locality. Meaning that as packets from the same flow become more dispersed, caching becomes more ineffective. Related to this is the fact that more micro-flows means more cache space (if the same level of service wants to be provided). Since line-speeds are increasing at a rate greater than that of memory size, caches can not be scaled to reach the growing demands of OC-192 links and beyond.

For these reasons, caching is not considered as a viable option for packet classification on network processors.

This chapter has introduced the fundamental principles and research that is applicable to network processors and packet classification. The issue of implementing packet classification algorithms on network processors, the focus of this research, will be dealt with in the next chapter.

Chapter 3

Algorithm Implementation

3.1 Introduction

The aim of this research is to test the suitability of network processors for classifying packets. The algorithms used for packet classification vary greatly in their design, it therefore becomes necessary to compare these algorithms in a standardised manner. This is achieved by using the same hardware and classification rule-sets for each implementation. This chapter will describe how each algorithm was implemented on the network processor hardware, as well as any modifications made to the algorithms in order to increase their performance.

3.2 Related Work

There have been several papers focused on implementing packet classification algorithms on network processors. However, these have been rather limited in terms of their scope. The papers differed from this research in that each implemented only one packet classification algorithm and no comparisons were made to determine the best suited algorithm. This related work is discussed below.

Ying-Dar Lin *et al.* focused on implementing the Diffserv architecture using the IXP1200 NP [35]. The implementation consisted of all tasks required by the DS edge-router, including multi-field packet classification. This was implemented using the BV algorithm, chosen due to its modest memory requirements and relative simplicity. Few details are

given regarding the actual implementation, though the authors do specify how the different functions are split up between the different MEs. The tasks of receiving packets, classification, policing and marking are performed in a single thread. Four of the six MEs were assigned to these tasks, while the other two MEs performed packet scheduling and transmission.

Srinivasan addressed a similar problem, however their work focused more on packet classification itself [57]. The hardware used was again the IXP1200, though now all the NP resources were being used for packet classification alone. This paper took a slightly different approach to the problem in that the focus was not on which packet classification algorithm to use, but rather on exploring how different design mappings of a particular algorithm can affect the performance of the implementation. The BV algorithm was implemented using two different design mappings. The parallel mapping assigned each incoming packet to a single hardware thread, which classified the packet completely. Each thread performs the same task in parallel. The pipelined design mapping breaks up the classification of a single packet between multiple MEs. Each ME performs the lookup for a single dimension and stores the partial result in SRAM before transferring the packet to the next ME in the pipeline. The two designs were compared and it was found that the parallel approach provided a better packet throughput. This is because the pipelined approach requires multiple SRAM accesses per packet, as packet headers need to be reread in each ME. This being the case, the parallel approach to implement the algorithm was used in our research.

The research of Tang *et al.* [63] used a propriety packet classification algorithm, similar to the Grid-of-Tries algorithm [60]. This paper provides interesting results in that a comparison of assembler and C-code implementations of the same algorithm is given. It was shown that the assembly language implementation improved the performance of classification by more than two fold. The reason is that the assembler code allowed for the number of SRAM accesses to be reduced by consolidating the data. This reduced the queueing delays incurred by multiple memory accesses and therefore increased the throughput. This same technique can be used with other algorithms and was adopted in our research.

An alternative programming language is the NP-Click architecture, which provides an abstraction to the hardware layer of the processor, allowing efficient code to be written quickly [50]. This programming model was shown to provide the same performance as an assembler-coded version of the same application, making it a viable alternative when programming NPs. It currently only supports the IXP1200 and was therefore not used in this research.

3.3 Implementation Considerations

Chapter 2.5 described the different packet classification algorithms. One of the key differences among these algorithms is that each heuristic-based algorithm exploits different properties of the classification database structure in order to increase performance. This structure was discovered by examining existing rule-sets and identifying their overriding structural properties. To compare various algorithms against one another, a standard rule-set must be used for all. Due to their private nature, there is a lack of large public-domain multi-field classification databases. The rule-sets obtained by the algorithm creators were sourced from large companies and ISPs [15, 59]. While randomly generated rule-sets can be used, they need to be generated in such a way that they contain the inherent structure present in real-world databases. There has therefore been research into creating a *packet classification benchmark* that provides the ability to create structured, artificial databases [66, 65]. This research uses the ClassBench [66] benchmark suite, proposed by Taylor and Turner. It is described in greater detail in Appendix A.

The implementation of the algorithms also has a upper bound in the amount of resources that they can use. The network processor used in the implementation is the Intel IXP2400. The key features of this NP are summarised below, further details can be found in Appendix B.

3.4 Intel IXP2400 Network Processor

The NP has 8 microengines that can be used to process packets. Attached to these microengines are a variety of memory technologies. These are listed, along with their specifications, in Table 3.1. While the SRAM can support a maximum of 128 MB, this is split over two independent channels. This allows both 64 MB banks to be used simultaneously.

The Radisys ENP-2611 evaluation board was used in our research. This is a PCI card containing an IXP2400 processor with 8 MB of SRAM and 256 MB of DRAM. It contains three Gigabit Ethernet ports that are used to send and receive packets [44]. The board runs the Montavista Linux 3.0 real-time operating system on its XScale host processor. This processor is responsible for initialising the MEs, but plays no role in the classification of packets.

Type	Size	Latency (cycles)
Local Memory	2560 bytes per ME	3
Scratchpad Memory	16 KB on die	60
SRAM	Max. 128 MB	150
DRAM	Max. 2 GB	300

Table 3.1: Available memory on the Intel IXP2400

3.5 Classification Databases used in Evaluation

Real-world packet classification databases are currently found in a variety of places, including Firewalls, VPNs and ACLs. Studies of these databases have shown that a rule-set contains between 10 and 5000 rules on average [66]; and this number is only expected to increase as QoS schemes become more common. In testing the different algorithms, it is therefore necessary to test various size rule-sets. The sizes of the rule-set used in this study is given in Table 3.2. The first column shows the number of rules that the ClassBench suite was requested to generate for each rule-set, the second column shows the actual number of rules generated. The numbers in these two columns are not equal due to the nature of the ClassBench algorithms. Once the requested number of rules has been generated, the algorithm removes any redundant rules – rules that are fully engulfed by a rule of higher priority. The effect of the difference in size is negligible as the order of magnitude of the rule-sets is still approximately the same[1]. Information regarding the exact parameters used in the generation of these rule-sets can be found in Appendix A.

3.6 Architectural Design

While the objective of this research is to test the suitability of network processors for packet classification, this only forms part of the process. The purpose of packet classification is to determine how a specific packet must be treated in a network element. If the network element is a firewall, then the classification will determine if a packet must be forwarded though the firewall or dropped. If the classification forms part of a DS-enabled router, it will be used (along with the information received from the metering unit) to mark the DSCP of the packet. For this reason, packet classification will firstly be discussed in how

[1]For consistency, rule-sets will be identified using the requested number rather than the generated number.

Requested No. of Rules	Generated No. of Rules
32	32
64	64
128	128
256	252
512	501
1024	1000
2048	1973
4096	3805
8192	7595
16384	14693

Table 3.2: The requested and generated rule-set sizes used in the evaluation

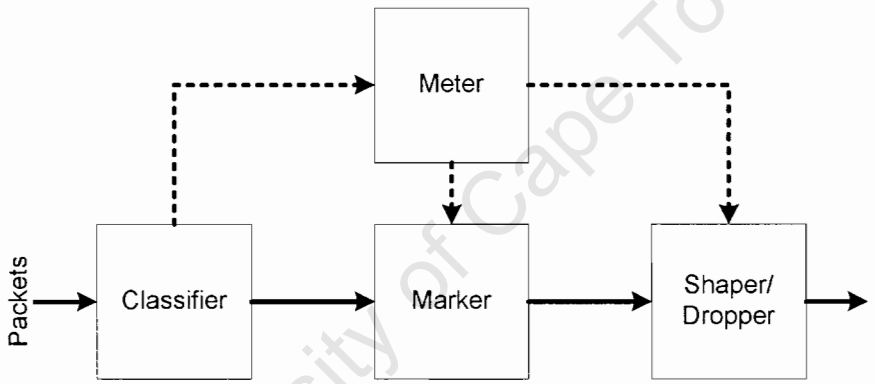

Figure 3.1: Functional structure of a Diffserv-enabled network node

it fits into the mapping of the complete DiffServ architecture to the NP, followed later by the mapping of the different packet classification algorithms specifically.

When moving from a design to an implementation, the algorithm must be mapped to the specific hardware available – in this case the NP. This *design mapping* will determine the final performance of the system and therefore it is important to make sure that the implementation will use the available resources efficiently [56]. Two main approaches can be followed when mapping an algorithm to the NP hardware: the pipelined approach and the parallel approach.

The architecture of a DS-enabled router was discussed in Section 2.3.1, including the functions that each router needs to perform: classification, marking, metering and shaping.

The interaction of these components can be seen again in Figure 3.1. To implement the above mentioned functionality in a NP using the *pipelined* approach, each function would be implemented in a different ME (Figure 3.2). This seeks to simplify the solution by breaking the design into a number of different tasks that can be performed individually. Packets will be received from the network interface by the first microengine then transmitted to the next engine for classification, which will in turn send the packet on for marking. One of the advantages of pipelining is that each ME only has to store the data structures that are used for that particular part of the functionality. For example, the classification ME needs only store the data structures used in classification and not those used by the metering and shaping units. There are a number of different mechanisms for transferring packets between microengines. The IXP2xxx series of NPs have a feature known as next-neighbour registers.

Next-neighbour (NN) registers can be used to transfer small amounts of data between adjacent[2] microengines with a latency of only 6 clock cycles. The problem with this method of transfer is that it must be performed in a synchronous manner. Once the data has been written to the NN-register, the producer ME must signal the consuming ME, to inform it that the data is ready to be read. This can be inefficient as the entire pipeline is slowed to the processing speed of the slowest unit.

The alternate method is to transfer data using the shared system memory. The IXP2400 has support for hardware-accelerated queues and ring buffers using the scratchpad and SRAM memories. These queues allow a producer to feed packets into a queue, which can then be emptied asynchronously by the consuming microengine. The downside of this approach is that even the fastest shared memory (the local memory in Table 3.1) has a latency of 60 cycles, which is much slower than the NN-registers.

While shared queues can be used in the straight pipelined approach, their usefulness becomes more apparent when considering the hybrid pipelined/parallel scheme discussed later.

In the *parallel* design (Figure 3.3), each packet is processed by a single microengine. The packet will be received from the network interface and processed completely, before it is transmitted to the outgoing network interface by the same ME. This approach achieves higher throughput speeds because multiple packets are processed in parallel, however several problems are introduced regarding memory accesses. If individual memory units are

[2]What is meant by 'adjacent microengines' is explained in more detail in Appendix B.

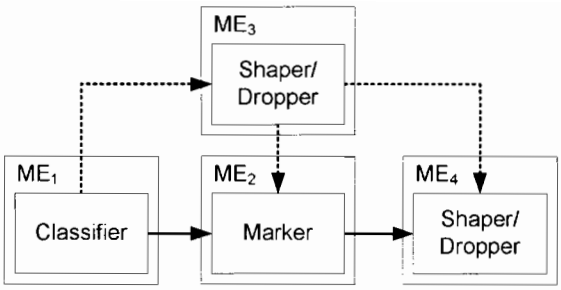

Figure 3.2: A DS-enabled network node implemented in a NP using the pipelined approach

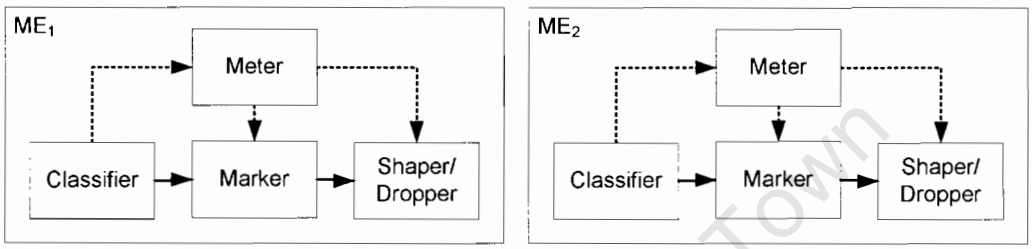

Figure 3.3: A DS-enabled network node implemented in a NP using the parallel approach

being used by each ME, all the required data structures will need to be cloned in each ME. This is inefficient, as data will be stored multiple times and some data structures – especially those that are used in packet classification – can be very large. There is also a problem with functions like packet scheduling and metering. These units require not only information regarding the current packet, but information regarding all packets that are currently being proposed or have been processed in the past. To ensure that this information is available to all units, it must be stored in shared memory, this introduces problems of synchronisation. A particular ME must ensure that no other MEs are busy using the shared data structures before writing to them. If this is not done, it is possible to introduce race conditions. This effect is not only present when accessing memory, but can happen with any of the shared resources in the NP. Another typical example of this can be seen with the output queues of the network interfaces. It will often be the case that two or more packets, arriving at different input interfaces, need to be transmitted on a single interface. This causes a similar synchronisation problem to the one seen above.

The *parallel/pipelined hybridisation* is the result of the observation that the different components of a network element will have different complexities. In the DS-enabled router example, the task of classifying and scheduling packets is generally more complex than

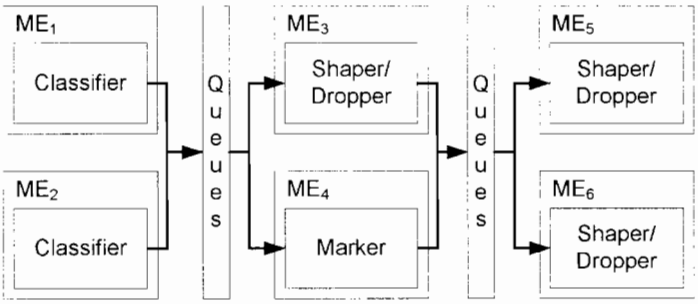

Figure 3.4: A DS-enabled network node implemented in a NP using the hybrid approach

the metering and marking stages. In order to maximise the throughput of the system, NP resources should be assigned proportionally, based on the complexities of the various stages. A possible design mapping for the example case in given in Figure 3.4. Here the classification and scheduling stages use twice as many microengines as the other stages. This design is also different from the traditional pipelined scheme in that queues must be used between stages. Since the complexity and therefore execution speeds of the stages are different, stages like the receiving of packets could conceivably process packets at twice the speed of the classification unit. It is therefore necessary to have the output of the receiver placed in a queue so that it can be emptied by the two classification MEs with packets being processed in parallel. These MEs must then place the data back in a queue so that the single metering engine can process each packet.

The hybridised approach combines the benefits of both the parallel and pipelined approaches, allowing for an efficient design that provides high throughput. This design is generally considered the best approach and is used in a number of different implementations [53, 30, 35].

3.7 Mapping the Algorithms to the NP

When performing the design-mapping of the packet classification unit, the same process needs to be followed. Each algorithm is unique in its requirements and must be individually examined to determine how the available resources are to be mapped. There are however several design decisions that apply to all algorithms, the choice of memory technology and the choice of processing unit.

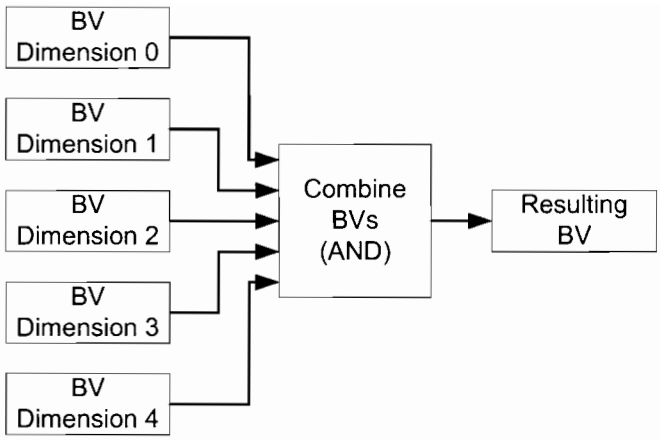

Figure 3.5: Parallel BV algorithm implementation

Of the different memory units available to the IXP2400[3], the only one that is suitable is the SRAM block. Both the local memory and the scratchpad memory are too small to hold the data structures of any of the algorithms. This was determined by examining the data structures created from the evaluation rule-sets in Table 3.2. While it is true that the data structures formed from some of the smaller rule-sets (i.e. 32 and 64 rules) are small enough to fit in the scratchpad memory, for the sake of consistency and implementation simplicity the larger SRAM was used for all rule-sets and algorithms. Instead it would have been possible to use the DRAM as it is both cheaper and larger. However, this design decision was rejected due to the slower access latency of this memory.

With regards to the choice of processing unit, the options were more limited. As discussed in Section 3.4, NPs have a number of PPEs, which are used for high-speed packet processing, and a control processor that is better suited for complex and non line-speed tasks. The intuitive design is to therefore use the PPEs[4] for the actual classification and the control (XScale) processor for the initial pre-computation stage of the algorithm. The complexity of the XScale comes at the tradeoff of speed, however this is acceptable as the pre-processing stage of the algorithms is only performed infrequently and does not need to be done at line speed.

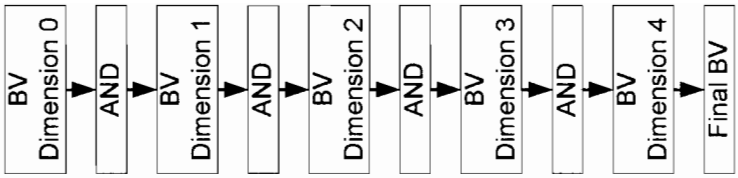

Figure 3.6: Serial BV algorithm implementation

3.7.1 Bit-Vector (BV)

The BV algorithm implementation suggested by the authors has a highly parallelised structure [31]. The design involved using separate SRAM units for each rule-set dimension. This allowed for each dimension to be fetched in parallel and then combined afterwards to yield the final result. This is portrayed graphically in Figure 3.5.

In order to port this same design to the NP, it would be necessary to interface with five individual SRAM units (one for each dimension of the classifier). Since the IXP2400 only supports two SRAM modules, this design needs to be altered. There are two main design decisions that need to be made: how many SRAM units should be used for the classification and secondly, how will the algorithm be split up amongst the MEs?

While the obvious answer to the first question would be to use as many memory units as available, in reality this is slightly more complex. As discussed earlier, packet classification almost always forms part of a larger system. Therefore, although it would be possible to use 100% of the NPs resources for the task of packet classification, it is necessary to provision resources for the other tasks as well. This led to the decision to use only one SRAM module for packet classification, leaving the other module free for tasks like packet scheduling and traffic metering, also both memory intensive tasks. Since the design is now limited to one memory module, the parallelism of the algorithm is reduced. Memory accesses must be performed in series leading to the design approach is shown in Figure 3.6. After the bit-vector for the particular range is returned, it is combined with the bit-vector from the previous range until the final bit-vector has been calculated.

Each incoming packet is assigned to an individual thread, which classifies the packet completely. This design has been shown to provide better performance than the pipelined approach where each stage of the algorithm is performed by a different ME [56]. Due

[3]These are given in Table 3.1.

[4]more commonly called MEs in the IXP2400

to the fact that the BV algorithm requires the bit-vector to have as many bits as there are rules in the classification rule-set, the size of the vectors can become very large. It is therefore necessary to process the vector in parts. Once the location in memory of the the bit-vector has been determined by the range-searching portion of the algorithm, the vector is processed in chunks. The size of these chunks is based on the width of the SRAM module – 4 bytes in the case of the IXP2400. Processing starts with the portion of the vector that contains the highest priority rules, as once a match is found, the rest of the vector need not be examined. The IXP2400 allows bit-vectors to be processed quickly as it has an instruction that will return the location of the most significant set-bit in a register. This allows the highest priority rule in a bit-vector to be found in one cycle.

3.7.2 HiCuts

The HiCuts algorithm takes a different approach to packet classification when compared to the BV algorithm. Instead of parallelising the task, HiCuts seeks to reduce the overall number of memory accesses. This approach is better suited to implementation on NPs, as the lack of multiple memory units does not hinder the design like it does for the BV algorithm. It would, however, be possible to split the decision tree over multiple memory units, therefore exploiting the independent nature of packet streams. This would allow different threads to access different branches of the tree in parallel, improving the throughput of the algorithm.

The processing stage of the algorithm computes the decision tree for a particular database and stores it in memory. The MEs will then classify packets by searching the decision tree until a match is found. This algorithm can be tailored in one of two ways. The first approach stores all node rule-sets as computed by the preprocessing stage (Figure 3.7). This ignores the fact that some rule-sets and rules will be duplicated in multiple leaf nodes and therefore consume more memory. The alternative is to instead store a list of references to rules (Figure 3.8). This insures that each rule is stored only once and the individual rule lists, in each leaf node, merely point to rules in this single set of rules. This approach minimises the storage requirements of the tree but introduces an additional memory access per rule. Both of these approaches are implemented and compared.

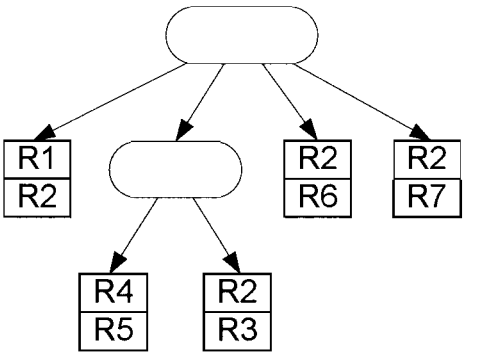

Figure 3.7: Direct rule lists used in the HiCuts algorithm

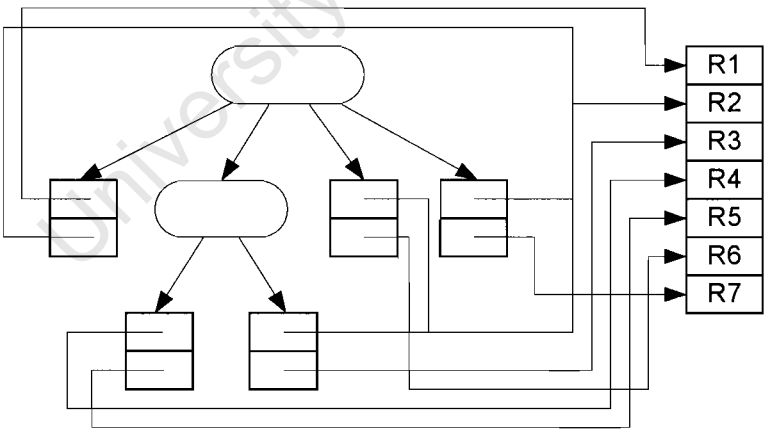

Figure 3.8: Indirect rule lists used in the HiCuts algorithm

3.7.3 Tuple

The algorithm that is implemented is the Pruned Tuple Search algorithm [59]. This works in much the same way as the BV algorithm, although instead of creating a bit-vector of rules, each bit in the vector now represents a tuple. This reduces the size of the vectors and therefore the storage requirements of the packet classification algorithm. However, there is the consequence that the priority encoding, which is present in the rule list, is now lost. This means that the highest priority rule can no longer be found in one cycle, instead each matching tuple must be examined in turn. There is also the added complexity of the hashing function. Although the IXP2400 contains a dedicated hashing unit that can compute hashes quickly, the indeterminate nature of hashing means that once the hash unit calculates the hash of a rule, the rule must then be compared to the incoming packet header, as it is possible that the rule does indeed not match and is the result of a hash table collision. This added complexity will negatively impact the classification speed, though the gained performance increase from the use of tuples should outweigh this.

3.7.4 Linear

The linear search algorithm has the most simple implementation. The rule-set is stored in SRAM and each packet classification thread compares the packet headers against the list of rules until a match is found.

Chapter 4

Evaluation Framework

4.1 Introduction

This chapter discusses the design of the evaluation framework that is used to test the performance of the packet classification implementations on the IXP2400. An important part of this process is ensuring that the test environment resembles real-world implementation environments.

4.2 Framework Requirements

The first requirement for creating the evaluation framework, is the need for realistic classification rule-sets. This research uses the ClassBench packet classification benchmark suite to generate rule-sets [66]. Various size rule-sets were generated, ranging from a minimum size of 32 rules to a maximum of 16384 rules. These sizes are similar to those seen in real rule databases [4]. The reason for using a benchmark suite, as opposed to a randomly generated rule-set, is that random rule-sets do not contain the inherent structure present in real-world databases. This is needed for heuristics based algorithms to function efficiently.

Secondly, the test traffic needs to be considered. In order to test the algorithm implementations, the framework needs to generate packets that can be classified by the network processor. Since each packet in the stream is classified individually, the more frequently the packets arrive, the faster the classification has to be performed. Therefore, to test the worst-case performance of the implementations, all packets in the stream should be

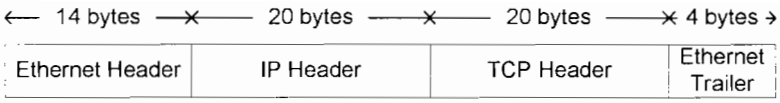

Figure 4.1: The Ethernet Frame and TCP/IP packet used for testing in the framework.

the smallest size. This will ensure the highest possible packet rate therefore giving the worst-case bounds for packet classification throughput.

While certain packet classification applications, like firewalls, would have very few rule-set matches when compared to the total volume of traffic, one would expect that a DS-enabled router rule-set would match a majority of traffic. This is true because rules refer to expected traffic streams, rather than malicious traffic. Rules in a Diffserv router are created in order to provide differentiated levels of service to applications that are in common use. This ratio of matching to non-matching packets will have different effects on different algorithms. For the linear search algorithm, a non-matching packet will cause the entire rule-set to be traversed. This will decrease the average packet throughput of this algorithm. In the case of the Tuple Space algorithm, there will be a positive effect, as the matching range of the packet will not contain any tuples that need to be compared with the packet headers. The scope of this study was limited to evaluating packet classification for Diffserv applications, therefore the test traffic is created such that every packet matches at least one rule in the classification database. The headers for these packets were generated by the ClassBench suite and then imported into the evaluation framework as 58-byte Ethernet-encapsulated TCP/IP packets (Figure 4.1). This is the smallest packet size supported by the framework and will therefore produce the highest packet rate. Different traffic streams were generated for each rule-set, with the number of packets being 10x the number of rules. This insures that each rule in the rule-set has approximately the same number of matching packets. These packets are then transmitted randomly to all ports of the NP at full line speed.

4.3 Architecture

The previous chapter described the mapping of the packet classification algorithm onto the NP hardware. For the evaluation framework this design is simplified slightly. Since the focus of this research is packet classification, this is the only functionality of the router that is implemented. The ME assembler code is based on the static forwarding sample application

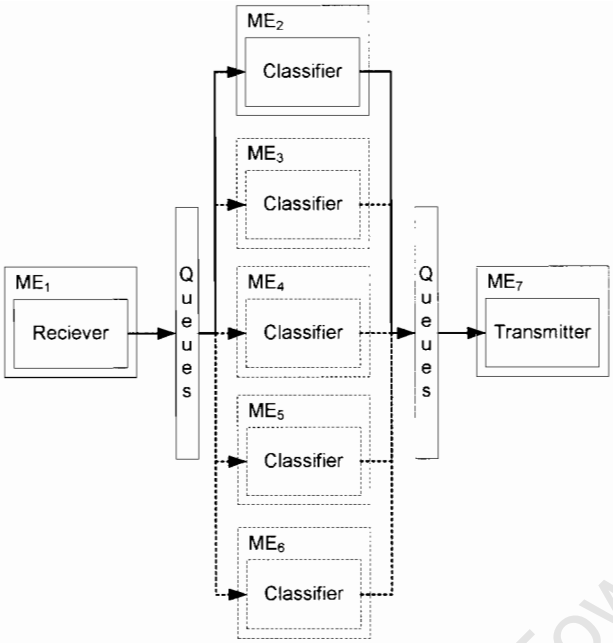

Figure 4.2: The ME implementation architecture for the packet classification evaluation framework.

that forms part of the IXP Software Development Toolkit (SDK). This application uses 3 MEs to forward packets between the ports of the NP. The first ME receives the packets from the network interfaces and places them in DRAM. Meta-data for the packet is placed in a queue in the faster Scratchpad memory. The meta-data describes the incoming and outgoing ports for the packet and the location of the packet body in DRAM. The second ME is responsible for polling the queue to determine if there are packets waiting. If this is the case, the meta-data of he packet is read and updated to indicate the packet's outgoing network port. The ME then copies the meta-data of the packet into the outgoing queue of the NP. This design allows the packet to be transmitted through the pipeline efficiently, without having to move the entire packet body[1]. The final ME then reads these packets from the outgoing queue and uses the meta-data to determine which port to transmit the packet on. The packet classification implementation works in much the same way, however instead of just redirecting the packets to different outgoing ports, packets are now classified as well. The packet headers are read from DRAM and placed into internal registers where they are then used in the various packet classification algorithms.

The preprocessing stage for each algorithm is written in C and is executed on the XScale

[1]The location of the packet in memory does not change, only the meta-data of the packet.

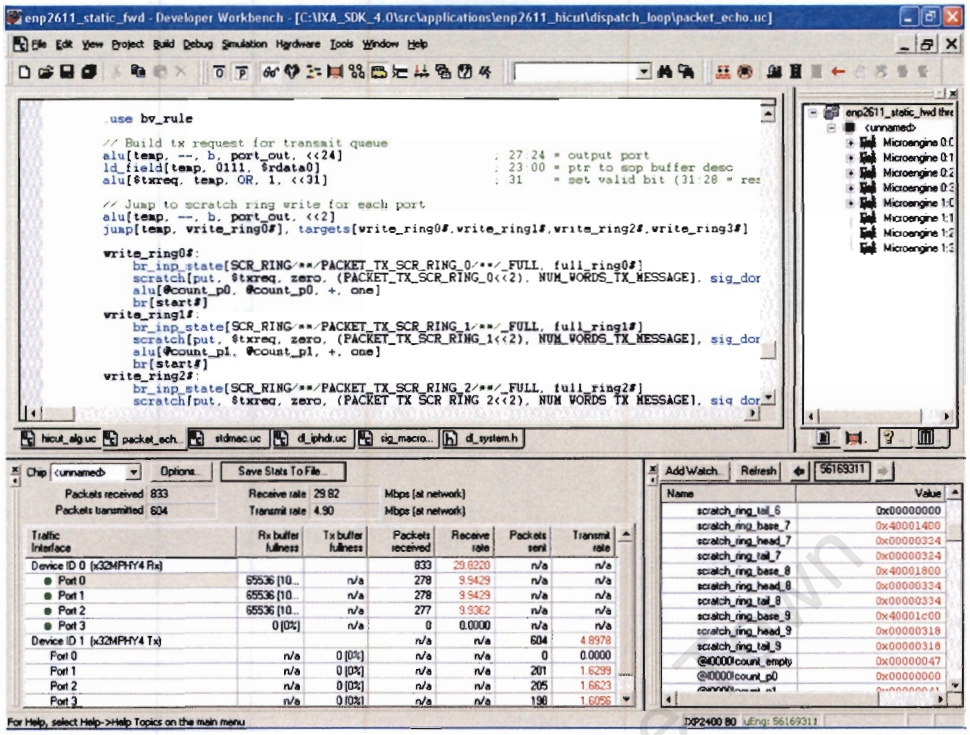

Figure 4.3: A screen shot of the IXP2400 simulator.

processor of the IXP2400. This stage imports the previously generated rule-set and creates the data structures needed by the classification algorithm. These structures are then stored in SRAM, where they can be accessed by the classification MEs. Since the framework uses queues to pass packets between the different MEs, it is possible to have multiple MES processing packets simultaneously. The implementation uses one queue for receiving packets and one for transmitting them. The classification stage using 1 to 5 MEs operating in parallel is shown graphically in Figure 4.2.

4.4 Simulation Environment

The structural model does not run on the actual NP hardware, but rather on a cycle-accurate simulator. This simulator is part of the IXP SDK and simulates both the MEs and peripherals on a clock-cycle by clock-cycle basis. The development environment also contains a packet generator that can be used to generate packets on the sending and receiving interfaces of the NP. The simulator runs exactly the same code as the hardware

NP, though it is better suited for evaluation and debugging purposes, as it allows a set of both internal and external metrics to be recorded. The external metrics recorded were the transmission and receiving rates of the network ports. The internal metrics, the fullness of the packet queues and memory utilisation, can only be measured by using the simulator. The IXP simulator runs on the Windows operating system and is part of the same development environment that is used for compiling and debugging ME code. A screen-shot of the environment is shown in Figure 4.3.

Each classification algorithm is implemented as a separate statically compiled process. These are then loaded into the a predetermined number of MEs during initialisation. The four algorithms were all tested individually, using the same rule-sets and input traffic. The results were obtained by running the simulation until a total of 10 000 packets had been received from all ports. This experiment was repeated five times for each algorithm with 1 to 5 MEs being used for classification. Metrics were collected for all four algorithms and for the modified HiCuts algorithm described in Section 3.7.2.

The raw data from the above mentioned experiments was collected and analysed, the results are presented and discussed in the next chapter.

Chapter 5

Results: Evaluation and Analysis

5.1 Introduction

This chapter evaluates the performance of different packet classification implementations within the NP evaluation framework. These metrics were collected from the experiments described in the previous chapter and are compared to determine the algorithms best suited to be mapped to the NP. The characteristics of these algorithms will also be discussed to determine which properties make some algorithms better suited than others.

The results presented in this chapter are a product of the metrics defined in Section 2.5.1. The algorithms will be compared using each metric separately. The final part of the chapter will then compare the algorithms as a whole, to determine which one is most suitable overall.

5.2 Storage Requirements

The memory requirement of an algorithm is an important metric, since this resource is limited in most network elements. In order to be able to use high-speed memory, the storage requirements of an algorithm must not be too large. The larger and faster the memory is required to be, the more expensive it is. Therefore the algorithms that perform the best according to this metric will be those with the lowest memory requirements. It is important to note that while this is the case, there is usually a tradeoff between the storage requirements and search speed of an algorithm.

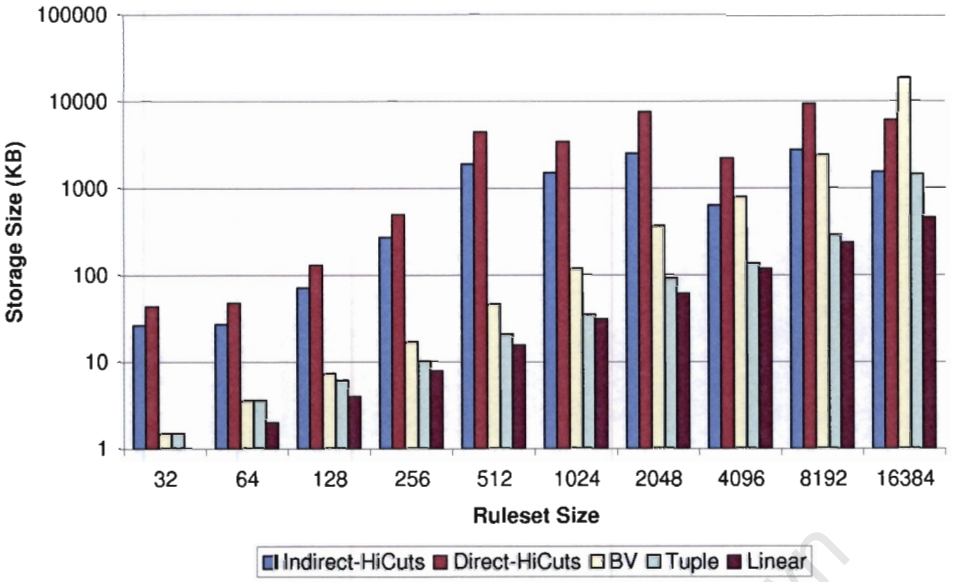

Figure 5.1: The memory space requirement for the data structures of different packet classification algorithms.

The storage metrics can be gathered from the preprocessing stage of the algorithms. When using a physical NP, the packet classification data-structures will be stored in SRAM after they have been calculated by the XScale processor. In the evaluation framework, the data structures are written to a script file which is executed during the startup of the simulator. This script copies the data into the virtual SRAM of the processor, allowing it to be accessed by the classification MEs during execution. Details regarding this process can be found in Appendix C.

The IXP2400 NP has a shared memory architecture, therefore increasing the number of classification MEs does not have an effect on the storage requirements of the algorithm. The results showing the memory requirements of the algorithm as a product of the rule-set size is shown in Figure 5.1.

Analysis

The simplest algorithm, the linear search, will always have an absolute value for its storage requirements. The only data structure for the algorithm is a list of all rules, requiring memory directly proportional to N, the number of rules in the set. The linear search algorithm has the smallest storage requirements for any algorithm in this study.

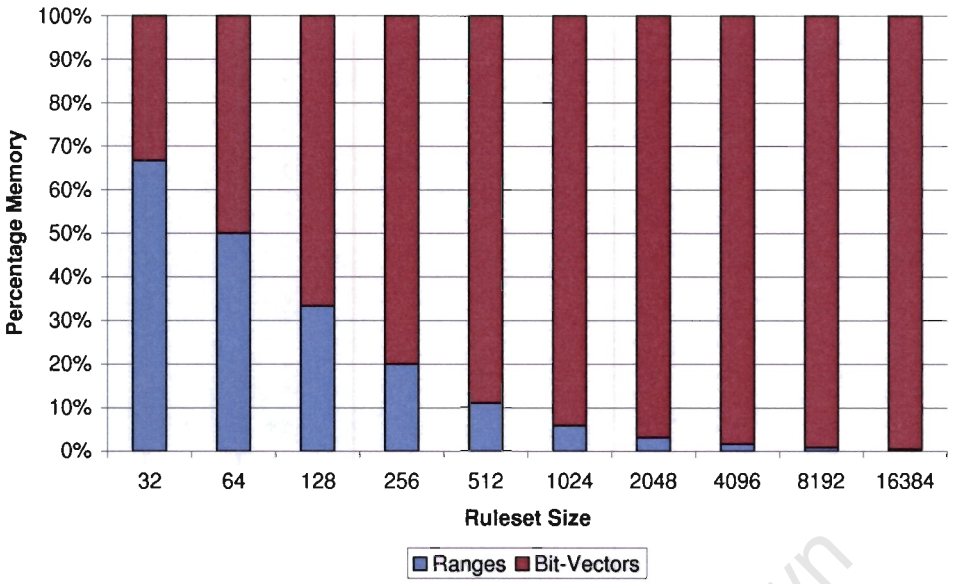

Figure 5.2: The storage requirement percentages for the two data-structures of the BV algorithm

The Bit-Vector (BV) and Tuple-Space Search algorithms have similar storage requirements. This is to be expected as both algorithms function in almost the same manner. These algorithms require two separate data-structures for each dimension of the classifier: a list of ranges and a bit-vector. The list of ranges is based on the input rule-set and is therefore identical for both algorithms. The BV algorithm has higher storage requirements for all cases, as the number of tuples is always equal to or less than the number of rules. There is a worst-case storage complexity of N^2 for the BV algorithm and NT for the tuple-space algorithm, T being the number of tuples.

The majority of the storage space for the two algorithms is used by the bit-vectors, this becomes more extreme as the size of the rule-set increases. Figures 5.2 and 5.3 show these ratios with regards to the different sized rule-sets for the two algorithms. The list of ranges is growing linearly while the bit-vectors grows quadratically for both cases.

The HiCuts algorithm differs from the other two algorithms in that its storage requirements are not proportional to the number of rules. The two largest rule-sets, for the experimental results, are for 8192 and 2048 rules. The 4096-rule set, which lies between the other two, is substantially smaller. The reason for this inconsistency is that the size of the HiCuts data-structures is less dependant on the number of rules and more dependant on the layout of the rules within the address space. If the rule-set has a large number of rules clustered

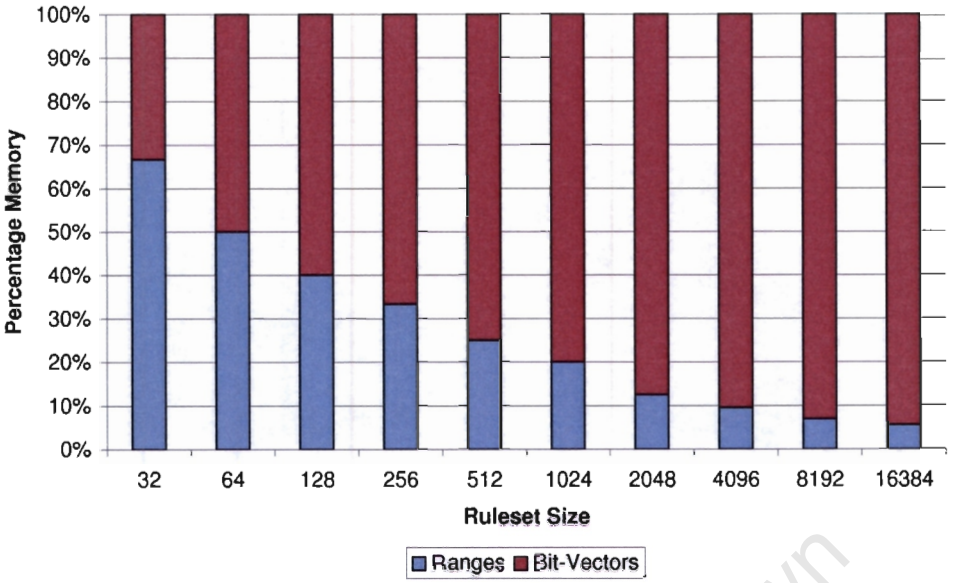

Figure 5.3: The storage requirement percentages for the two data-structures of the Tuple-Space algorithm

in small areas, the algorithm will have a larger number of rules in each leaf node and a shallower tree[1]. This will decrease the storage requirements for the rule-set. If the rules are more evenly distributed throughout the address space, each node will contain more children and it is more likely that rules will be duplicated in adjacent nodes. This will increase the storage requirements for the rule-set. It is difficult to determine the size of a HiCuts tree without first creating it, however a worst-case bound of N^d, d being the number of dimensions, is given in the literature [16].

The Indirect and Direct HiCuts algorithms differ quite dramatically in regards to their storage requirements, this becomes more apparent as the size of the rule-set increases. For the 16384-rule set the Direct-HiCuts version requires almost 4 times as much memory as the indirect version.

5.3 Search Speed

The second metric is the search speed of the algorithm. In order for an algorithm to be useful, it needs to be able to classify packets at realistic network line-speeds. The IXP2400

[1]See Section 2.5.3.

can forward packets at a maximum speed of approximately 4400 Mbps, this is therefore the best performance that can be expected from a packet classification algorithm.

For most processors, there is a large discrepancy between the time taken to execute a general instruction, and the time taken to access external memory. In the IXP2400, it takes 150 cycles to access SRAM as opposed to 1 cycle to execute an instruction. Packet classification is a data-intensive task and will therefore have an execution time that is dominated by the number of memory accesses required. This section will therefore use memory accesses as an indication of the search-speed of the algorithms.

The IXP2400 uses multiple MEs and hardware threading to hide the latency of memory accesses. This allows the NP to process a number of packets in parallel. This design does not increase the processing speed of individual packets, but rather the overall packet throughput. The following results show the total transmission speed for packets on the outgoing ports of the NP. This metric was used, rather than the receiving rate of incoming packets, as it more accurately represents the speed at which classification takes place. Incoming packets are queued before they are classified, this allows them to be received at a higher rate than they can be processed at. The transmission rate of the NP is measured at the outgoing network port. This metric is recorded by the simulator and is the basis for the following results. More details regarding how the metrics are obtained from the simulation environment are given in Appendix C.

This section will firstly consider each algorithm separately, comparing the results from using different numbers of MEs for classification. The algorithms will then be compared against one another to determine which algorithm provides the best performance.

5.3.1 Linear Search Algorithm

The linear search algorithm is the most straight forward method for classifying packets and therefore high performance is not expected. However, these results will show the lower bound of the problem and create a platform on which the other results can be considered. As mentioned earlier, the upper bounds of the system are obtained by considering the case where packets are not classified and only forwarded. This will show the constraints of the hardware and the other two MEs in the pipeline.

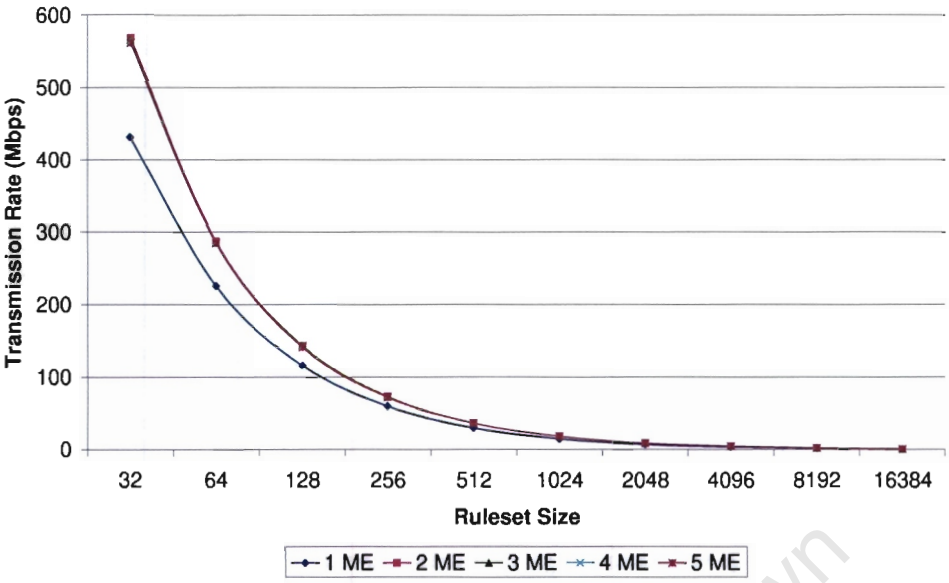

Figure 5.4: The packet throughput for the linear search algorithm.

Analysis

The results of the linear search are given for all rule-sets, using a number of MEs, in Figure 5.4. For this algorithm, the performance decreases as the number of rules increases. On average, $\frac{N}{2}$ memory accesses are required per packet.

Notice also that while the performance does increase as the number of MEs is increased, it does not do so linearly as would be expected. This is the case for all the algorithms in this study and will therefore be discussed separately in Section 5.3.4.

5.3.2 Bit-Vector (BV) and Tuple-Space Algorithms

The search-speed of the BV and Tuple algorithms is dependant on two parts. The first step is finding the matching range. This needs to be done for all 5 dimensions of the classifier, with a theoretical maximum of $2N + 1$ ranges per dimension. The ranges are stored in sorted lists and it is therefore possible to use the binary search algorithm to find the enclosing range of a packet with $O(\log N)$ memory accesses. The bit-vectors for the BV and Tuple algorithms need to have as many bits as there are rules and tuples respectively. Table 5.1 shows the number of tuples in each of the rule-sets used in this research. This agrees with the results obtained by the tuple-space algorithm authors: the rate at which

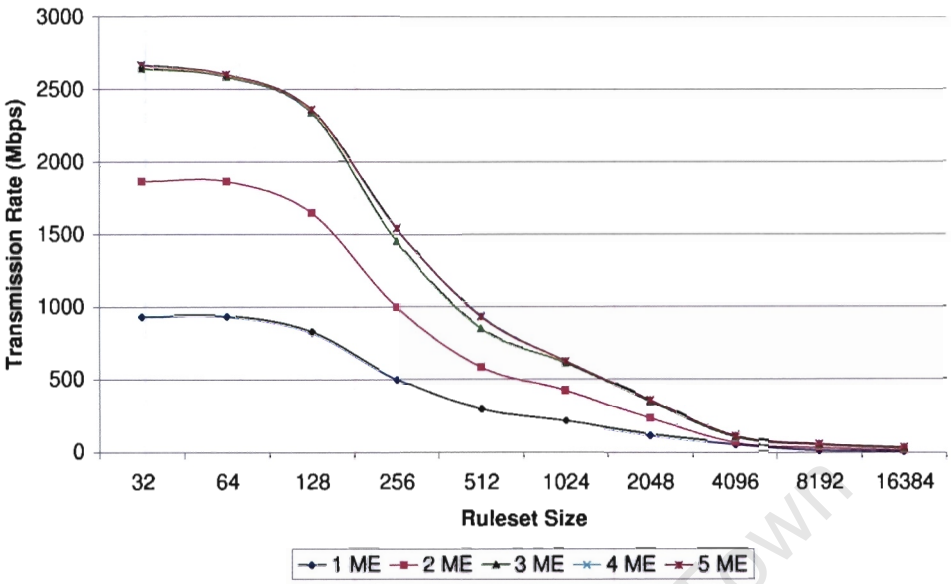

Figure 5.5: The packet throughput for the BV algorithm for multiple MEs

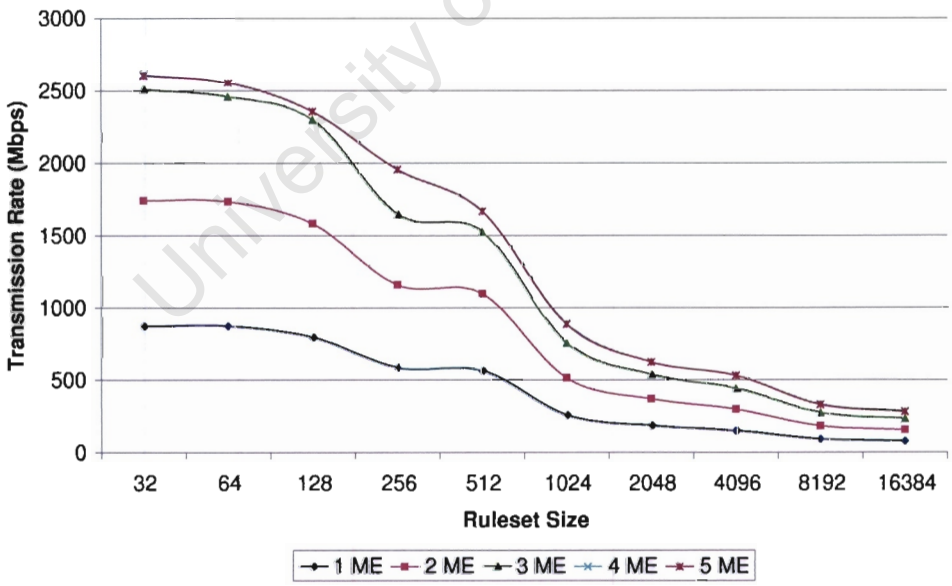

Figure 5.6: The packet throughput for the Tuple-Space algorithm for multiple MEs

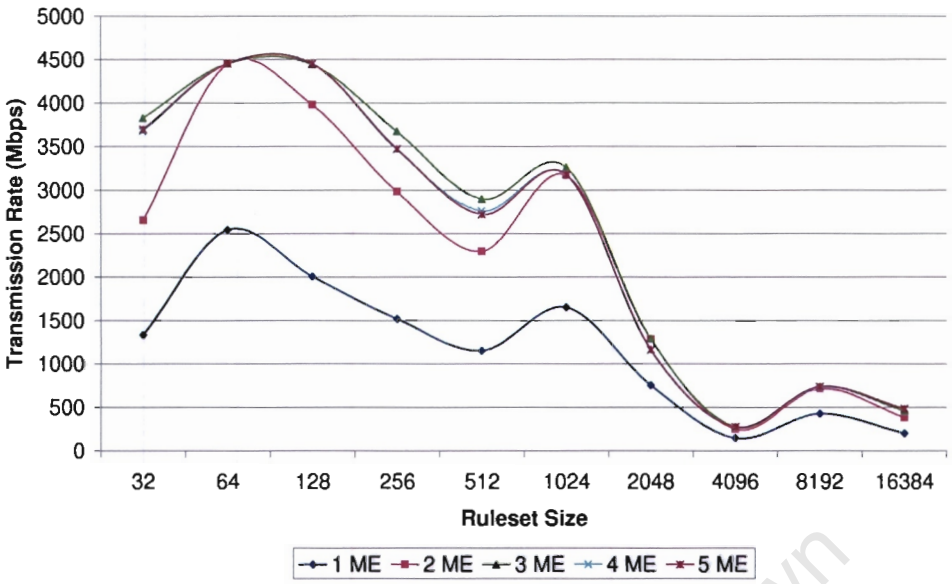

Figure 5.7: The packet throughput for the Direct-HiCuts algorithm

5.3.3 HiCuts Algorithm

The performance metrics for the HiCuts algorithm are more difficult to analyse than the other algorithms in this study. Figures 5.7 and 5.8 show the search-speed metrics obtained for both versions of this algorithm. While the performance of the other algorithms decreased as the size of the rule-set increased, this is not the case for the HiCuts algorithm. The highest packet throughput for this algorithm is seen for the 64-rule set and the lowest for the 4096-rule set. To understand why this is the case, the underlying structure of the rule-set decision-trees needs to be examined.

Analysis

Like the other algorithms in this research, the classification performance for the HiCuts algorithm is dependant on the number of memory accesses per packet. The memory accesses for this algorithm are split between two processes: the tree traversal and the rule comparison. It requires 1 memory access to traverse each node, therefore the maximum number of memory accesses is equal to the maximum depth of the tree. A more realistic metric is given by examining the average depth of the tree. This is determined by taking

[2]The width of the SRAM is 32-bits.

[3]This assumes each tuple is assigned to a unique hash key and therefore no conflicts occur.

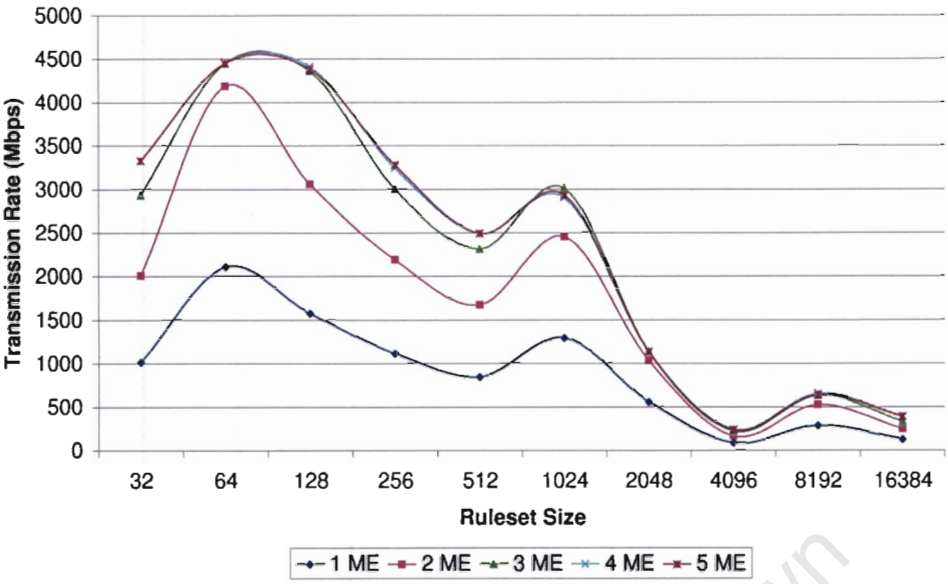

Figure 5.8: The packet throughput for the Indirect-HiCuts algorithm

the total depth of all leaf nodes and dividing it by the total number of leaf nodes, the results of this are shown in Table 5.2. The average depth will give an indication of the average number of memory accesses required to reach a leaf node. The algorithm must then compare each rule in the leaf node with the current packet headers. The maximum number of rules in a leaf node is determined by the *binth* parameter, which in turn is determined by the structure of the rule-set [48]. The average number of rules in a node is considerably less than the maximum (Table 5.2). While there are a few nodes with approximately *binth* rules, the majority contain very few. Each rule comparison requires 2 memory accesses with an average of $\frac{N_L}{2}$ rules per packet, N_L being the number of rules in the particular leaf node. The overall performance for the HiCuts algorithm would therefore be proportional to $d_A + \frac{N_A}{2}$, where d_A is the average depth of the tree and N_A is the average number of rules per leaf node. This is show in Figure 5.9.

The point is illustrated further in Figure 5.10. The rule-set in (a) consists of 5 rules and has a maximum depth of 3. In order to match rule R2, a total of 7 memory accesses is required, 3 for the node traversal and 2 for each rule. Tree (b) has double the rules of Tree (a) but is only 2 levels deep. This case requires 1 memory access to reach the first list of rules and another 4 to match rule R2, a total of 5 memory accesses.

No. of Rules	Max. Depth	Avg. Depth	*binth*	Max. Rules	Avg. Rules
32	9	3.8	4	4	2.3
64	7	2.6	4	4	2.4
128	10	2.8	4	4	2.6
256	9	3.1	8	8	3.5
512	11	3.8	8	8	2.4
1024	9	2.9	16	16	3.3
2048	9	3.5	32	32	3.7
4096	3	1.6	256	250	8.8
8192	8	2.6	64	64	6.1
16384	4	1.1	512	512	9.6

Table 5.2: The average depth and number of rules in the rule-sets.

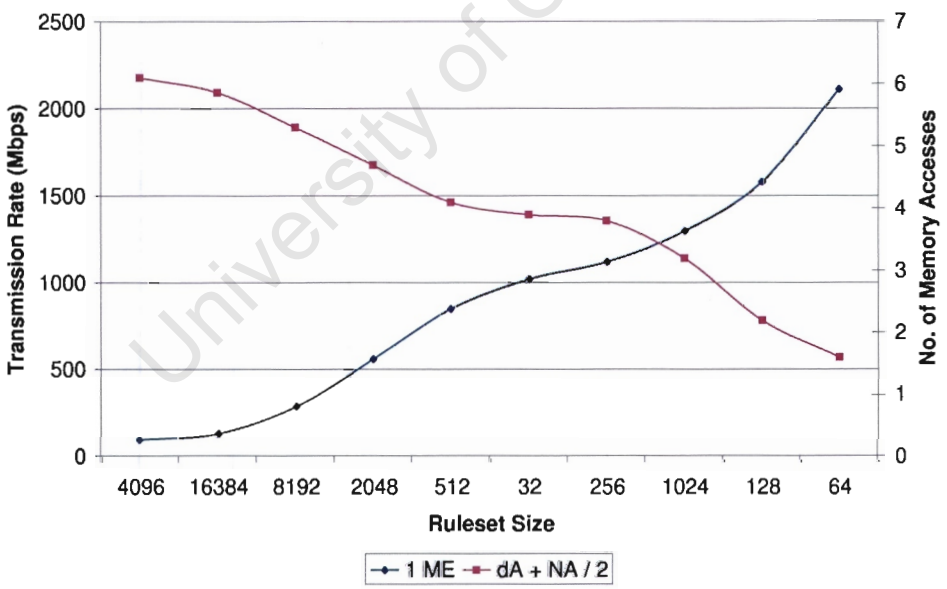

Figure 5.9: The performance of the HiCuts algorithm with respect to the average tree depth and number of rules per node

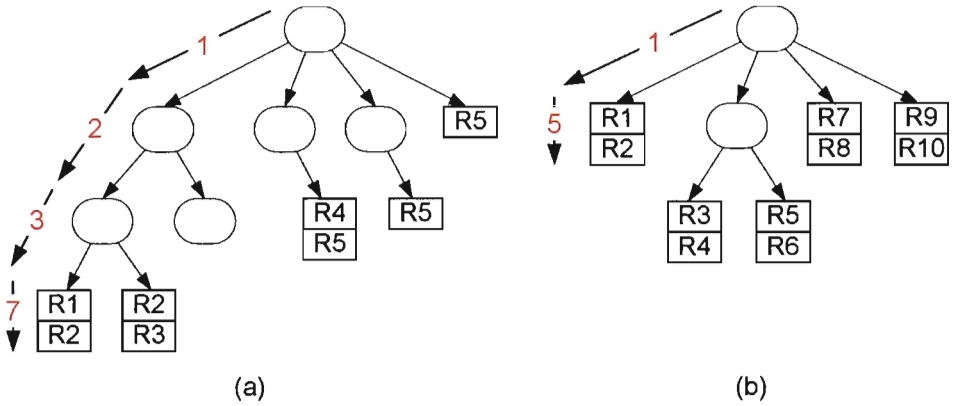

Figure 5.10: An HiCuts tree showing the number of memory accesses required for (a) an example 5-rule set and (b) an example 10-rule set.

5.3.4 Performance Increase from using Multiple MEs

This section will address the issue of using multiple MEs for classification. All the algorithms discussed above show a performance increase as more MEs are added to the system, however as expected this increase is not linear. This is due to the two bottlenecks in the system: processing power and memory utilisation.

During classification there will be a number of threads accessing the memory simultaneously. The SRAM controller has a number of queues in which memory requests are placed until they can be executed. The NP was designed in this way, to hide the memory latency by allowing other threads to execute, while some are waiting on memory operations. This has the advantage of allowing multiple packets to be processed in parallel, therefore increasing the memory throughput.

Memory accesses can however only be performed at a finite rate and therefore as the number of MEs increases, so does the memory utilisation. Once this reaches 100%, no more memory accesses can be queued and increasing the number of MEs will have no effect on the packet processing speed. Figure 5.11 shows this graphically for the HiCuts algorithm using the 8192-rule set. Related to this is the fact that increasing the number of MEs once the memory has been totally utilised can have a negative effect on the transmission rate, this can been seen in Figure 5.11 when increasing the number of MEs from 4 to 5. This effect is caused by the SRAM command queue back pressure mechanism [63].

Also worth noting in this section are the results for the 64 and 128 rule sets. For 4 and 5 MEs, the classification performance is no longer bounded by memory, but rather by

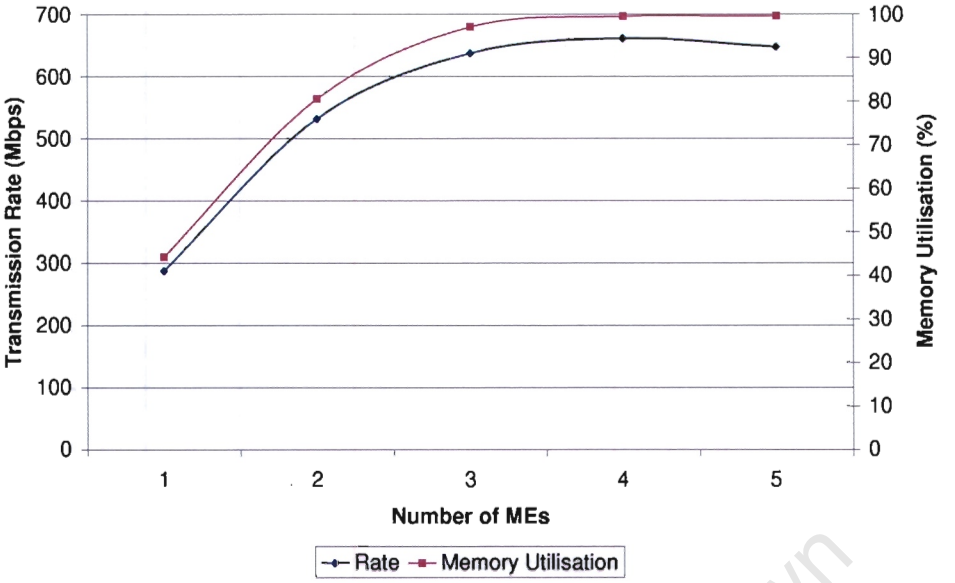

Figure 5.11: The increased performance and memory utilisation as the number of MEs is increased.

the other parts of the architecture, namely the receiving and transmitting MEs and the physical network ports of the processor.

For all the algorithms, the maximum transmission rate was reached using 4 MEs. At this stage, memory utilisation is close to 100% and adding additional MEs did not increase the performance. For the 64 and 128 rule sets, the transmission rate is not limited by the memory utilisation, but rather by the maximum limitations of the hardware. The maximum rates for all the algorithms are summarised in Figure 5.12.

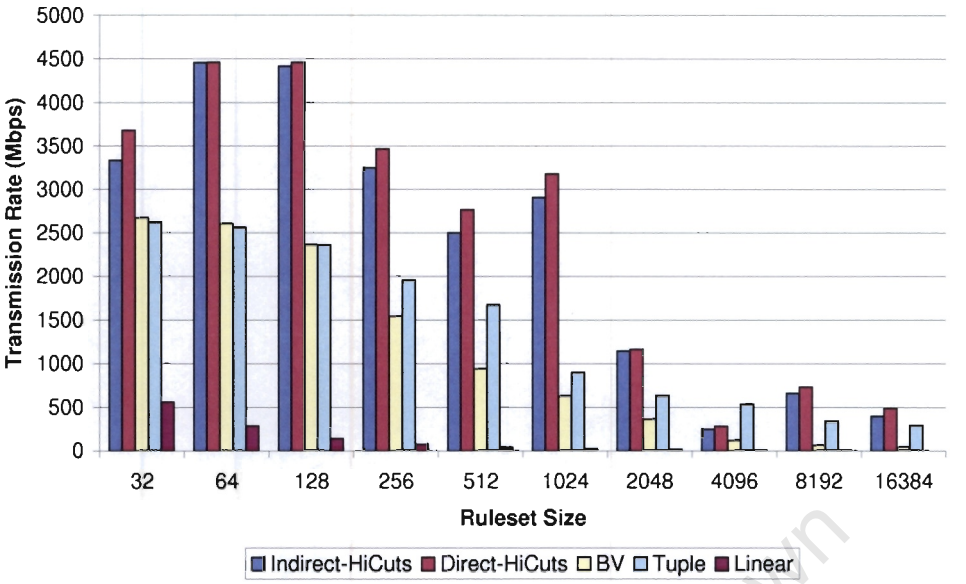

Figure 5.12: The maximum transmission rate for all the algorithms using 4 MEs

5.4 Algorithm Latency

Another important metric, which is related to the search-speed, is the latency introduced by the packet classification algorithm. Real-time applications, like VoIP and video conferencing, are particularly affected by this. In this research, latency is defined as the time taken, from beginning to end, to classify a packet.

The packet throughput rate of an algorithm measures the total number of packets classified per second. This metric can be increased by processing more packets in parallel. Latency is measured on a per packet basis and therefore can only be improved by modifying the classification algorithm. The results shown in Figure 5.13 are a measure of the average latencies for the different algorithms. They were obtained by adding code to the packet classification MEs that recorded the elapsed number of cycles before and after the algorithm had executed. The difference between these two values can be used to calculate the latency of the algorithms. More information regarding this can be found in Appendix C.

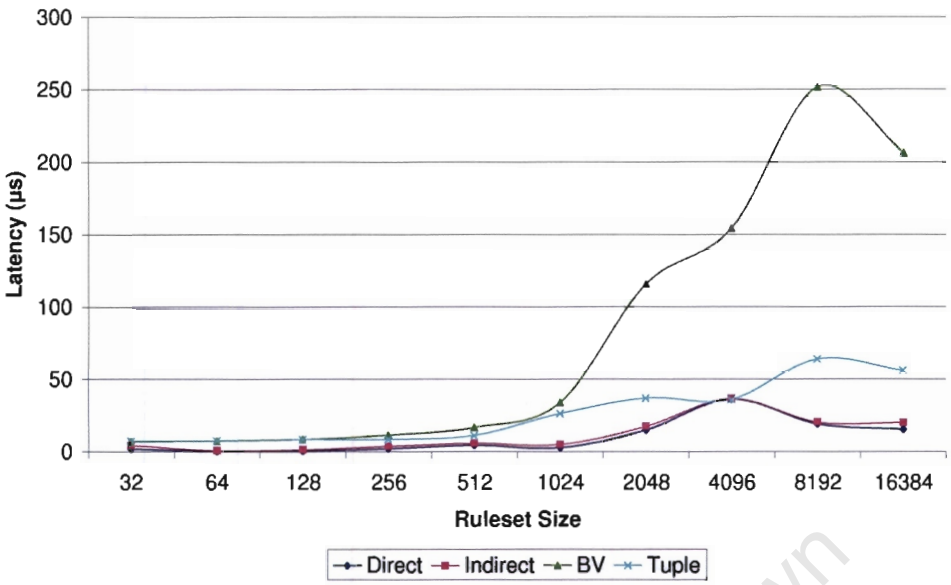

Figure 5.13: The latency of the different packet classification algorithms as a function of the rule-set size.

Analysis

The lower the latency, the more packets that can be classified per second and therefore the higher the packet throughput of the algorithm. The packet throughput is therefore inversely proportional to the latency, this can be seen in Figure 5.14.

The highest latency for all rule-sets and algorithms is 251 μs. This number is small when compared with the 150 ms delay that is acceptable for VoIP traffic. This architecture is therefore suitable as a classification system for real-time applications. The algorithm which performed best in terms of latency was the HiCuts algorithm.

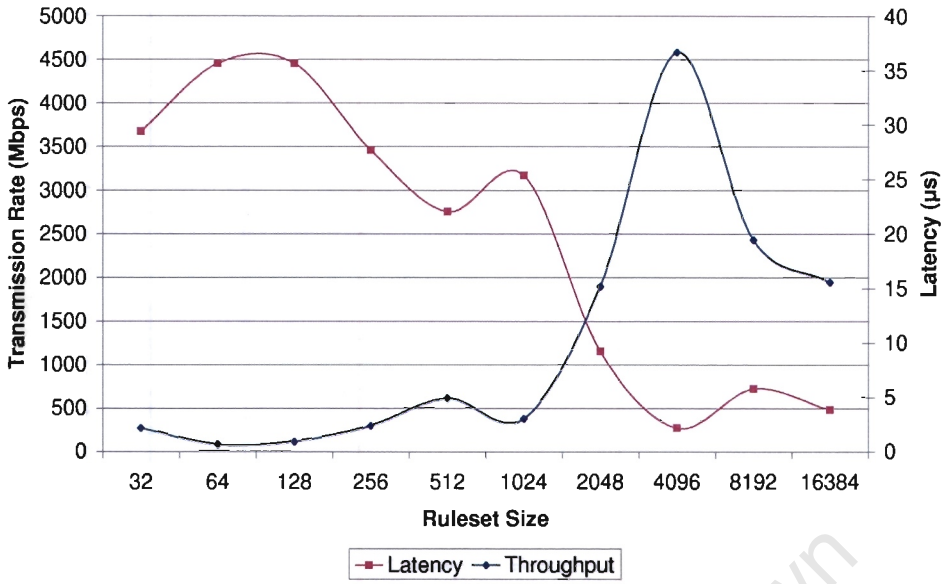

Figure 5.14: The latency and packet throughput of the HiCuts algorithm using 4 MEs.

5.5 Summary

In comparing the different packet classification algorithms, there are a number of observations that can be made:

Firstly, the algorithm that performs best with regards to classification speed is the Direct-HiCuts algorithm. This is true for all the rule-sets used in this research, save one: the 4096 rule-set. The low performance of this particular rule-set is not a product of the number of rules, but rather due to the structure of the rules. This makes it difficult to gauge the performance of this algorithm for a particular rule-set, without first performing the preprocessing stage of the algorithm.

In the case of the 4096-rule set, where there are a large number of clustered rules, the algorithm that performed best was the tuple-space algorithm. This algorithm had the second highest overall performance (after the two HiCuts variations) and outperformed the BV algorithm in all cases except the two smallest rule-sets. For these two rule-sets, the reduction in the size of the bit-vector from using tuples was not large enough to offset the added complexity of the algorithm. Due to the difference in growth rates between the number of rules and the number of tuples, as the rule-set size becomes larger, the benefits of the tuple-space algorithm should become more pronounced.

In terms of storage requirements, the algorithm with the best results is the linear search algorithm. However, this algorithm is not considered due to its poor performance with regards to search-speed. The worst overall performer is the Direct-HiCuts algorithm, with large memory requirements for even the smaller rule-sets. As the size of the rule-sets increases, the difference between the algorithms becomes less extreme. In fact, the growth rate of the BV algorithm data-structures results in it requiring more space than the HiCuts algorithm for the 16384-rule set. This, along with slower rate of increase for the HiCuts algorithm, would suggest that as the number of rules increases further, the difference between the storage requirements of the algorithms would decrease even more. The largest data-structure encountered in this research was for the 16385-rule set using the BV algorithm, with a total size of 18 MB. This number is however twice as large as the second largest rule-set, which is 9 MB. Both of these data-structures are still smaller than the maximum supported SRAM size per channel of 64 MB.

This research has shown that while not all packet classification algorithms are ideally suited for the architecture of a NP, it is possible to achieve reasonable results doing so. Heuristics-based algorithms must be used to provide reasonable performance. The linear search algorithm employs an exhaustive comparison of the rule-set and has a search-speed far inferior to the other algorithms.

The performance of heuristics-based algorithms will depend largely on the structure of the rule-sets being used for classification. NPs have the advantage in this regard as since they are software programmable, various packet classification algorithms can be used with the same hardware. The architecture would only need to examine the classification rule-set and load the MEs with the most suitable algorithm. This flexibility and the results of this research identify network processors as being highly suitable for packet classification.

Chapter 6

Conclusions and Recommendations

6.1 Conclusions

The main objective of this study has been to determine the suitability of the network processor architecture for packet classification. Network processors provide a high-speed platform for network-processing applications. This, together with their high degree of flexibility makes them a logical choice for the complex task of packet classification.

In examining the architectural design of NPs and the different mechanisms used by current packet classification algorithms, it was found that there are two approaches used to reduce the complexity of packet classification: reducing the number of memory lookups per packet and increasing the parallelism of the design. These algorithms were then mapped to the NP architecture and an evaluation framework was developed around this.

The framework was designed to emulate the conditions of a DiffServ network element and the experiments followed this model. Metrics regarding the search-speed and storage space of the algorithms were recorded and analysed to give the results provided in the previous chapter. The following conclusions have been drawn from these results:

- As complex network applications become more popular there is an increasing need for more intelligence in the network elements themselves. An important requirement for this is the ability to differentiate between different classes of traffic. This has led to a great deal of research into multi-field packet classification. Unfortunately, packet classification is a complex operation with poor worst-case time and storage bounds.

This is especially true when compared with other advanced network functions, like packet scheduling and buffer management. Current algorithms have therefore used a heuristics-based approach to simplify the problem.

- Existing classification schemes have focused on solving this problem algorithmically. Little attention has been given to the architecture on which the algorithms will be implemented, with most research evaluating software or custom-hardware based versions of the algorithms. Thus, there is a need for an evaluation of the different packet classification schemes using existing network processors.

- Network processors provide a flexible architecture on which this design mapping can be done. The high packet processing speeds and support for complex operations make them a suitable target platform for packet classification.

- Previous research on using network processors for packet classification has been limited, and has either proposed custom packet classification hardware units or has only tested a specific algorithm on existing NPs. No research has been done on comparing different packet classification algorithms to determine which provide the best performance when mapped to existing NPs.

- The mapping of the algorithms to the NP hardware is an important design decision that will have a large effect on the final performance of the system. It is therefore necessary to consider the system requirements of each algorithm and how these can be fulfilled most efficiently by the NP.

- Programming NPs is a complex task, which requires a good understanding of the processor's custom instruction set and underlying architecture. The large degree of parallelism complicates matters further. Research has started to focus on this problem, with architectures like NP-Click[10] that provide a high-level model specialised for NPs, while still providing performance similar to that of traditionally programmed systems.

- The evaluation framework used in this research has shown that it is possible to map a variety of algorithms to the NP, with varying levels of performance. Indeed, the algorithms which provided the best overall results were those which required the fewest number of memory accesses per packet – this being the primary bottleneck of the system. The other bottleneck, processing power, could be overcome by increasing the number of MEs used for classification.

These conclusions, along with the results described in the previous chapter, lead us to believe that current packet classification algorithms can be efficiently mapped to the NP. The performance achieved by these mappings make them sufficient to satisfy the conditions introduced in Chapter 1. Furthermore, by using improved hardware and packet classification algorithms it is possible to achieve better results than those previously obtained in similar studies [56, 35].

6.2 Recommendations for Future Work

Packet classification and NPs are both areas of research in which there are still many avenues that have not yet being explored. This research is by no means a comprehensive study and there are a number of issues that warrant additional research. The recommendations for future work that have stemmed from this study are presented below:

- It would be possible to extend this research by performing the same study using additional algorithms. There are a large number of packet classification algorithms, and while this study sought to deal with a few that would represent the entire spectrum, it is possible that there are other algorithms that will perform better than the ones in this study.
- Following on from the above point, modifications for many of the algorithms used in this study have been suggested in literature [4, 52, 58]. These could be tested against the original algorithms in order to determine the gained performance.
- This research focused on evaluating the packet classification algorithms according to the metrics of search-speed and storage requirements. These metrics are particularly important for the DiffServ architecture, however the other metrics of updatability and preprocessing time might be more important for other applications, like IntServ, where the rule database is less static. It would therefore be of interest to test the NP design mapping of the algorithms with these metrics. The preprocessing stages used in this research were designed with simplicity, rather than performance, in mind. It would therefore be relatively easy to improve on these.
- The Intel IXP2400 is the only NP used in this research. While most NPs share a fairly similar architecture, it is possible that some might be better suited to the task

of packet classification than others. This research could be extended by broadening the scope of the study to focus on the relative performances of different NPs.

- The algorithms which reduced the number of memory accesses performed better than those that sought to parallelise the classification, as most of this parallelism was lost when mapping the algorithm to the NP. For future algorithms using current NP hardware, the less memory accesses required to classify a packet, the better the overall performance of the design.

- The task of packet classification can be easily parallelised, this should be taken into account when designing future NPs. The multiple MEs in a NP take advantage of the parallelism. However, the number of shared memory units is far fewer than the number of MEs, this forces memory requests to be processed sequentially. Having a number of memory units operating in parallel will allow multiple memory requests to be processed simultaneously. Current next-generation NPs seem to offer improved processing power and additional hardware for specialised tasks, while offering little improvement of the memory architecture. As this was the primary bottleneck in this research, it is an important area to consider. One NP that does take this into account is the IXP2800, which improves on the IXP2400 by introducing an additional 2 SRAM channels and 2 DRAM channels.

- While there has been limited research into programming models for NPs (e.g. NP-Click), this is still an area with a large scope for improvement. Increasing the ease with which NPs can be programmed will speed up the adoption of this technology. A unified programming model that supports a number of different NPs would allow developers to focus more on the functionality of the device and less on the underlying hardware.

Bibliography

[1] Janet Abbate. *Inventing the Internet.* MIT Press, 1999.

[2] P. Almquist. Type of Service in the Internet Protocol Suite. RFC 1349, July 1992.

[3] Florin Baboescu, Sumeet Singh, and George Varghese. Packet Classification for Core Routers: Is there an alternative to CAMs. In *INFOCOM*, 2003.

[4] Florin Baboescu and George Varghese. Scalable Packet Classification. *IEEE/ACM Trans. Netw.*, 13(1):2–14, 2005.

[5] Y. Bernet, S. Blake, D. Grossman, and A. Smith. An Informal Management Model for Diffserv Routers. RFC 3290, May 2002.

[6] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Service. RFC 2475, December 1998.

[7] R. Branden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP). RFC 2205, September 1997.

[8] B. Caprita, W.C. Chan, and J. Nieh. Group Round-Robin: Improving the Fairness and Complexity of Packet Scheduling. Technical Report CUCS-018-03, Columbia University, June 2003.

[9] Computer Emergency Response Team Coordination Center. E-Crime Watch Survey. Survey, May 2004.

[10] Yie-Tarng Chen and Shin-Shian Lee. An Efficient Packet Classification Algorithm for Network Processors. In *IEEE International Conference on Communications*, pages 1596–1600, May 2003.

[11] Economics and Statistics Administration. United States Department of Commerce: E-Stats. U.S. Census Bureau Survey, May 2005.

[12] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. In *IEEE/ACM Transactions on Networking*, 1993.

[13] Bob Gelinas, Paul Alexander, Charlie Cheng, W. Patrick Hays, Ken Virgile, and William J. Dally (Lexra). NVP: A Programmable OC-192c Powerplant. Presentation, Network Processor Forum, June 2001.

[14] P. Gupta and N. McKeown. Packet Classification using Hierarchical Intelligent Cuttings. In *Hot Interconnects*, August 1999.

[15] Pankaj Gupta and Nick McKeown. Packet Classification on Multiple Fields. In *Proceedings of SIGCOMM*, pages 147–160, 1999.

[16] Pankaj Gupta and Nick McKeown. Algorithms for Packet Classification. *IEEE Network Special Issue*, 15(2):24–32, March 2001.

[17] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. Assured Forwarding PHB Group. RFC 2597, June 1999.

[18] J. Heinanen, Telia Finland, and R. Guerin. A Single Rate Three Color Marker. RFC 2697, September 1999.

[19] J. Heinanen and R. Guerin. A Two Rate Three Color Marker. IETF RFC 2698, September 1999.

[20] A. Heppel. An Introduction to Network Processors. Technical report, Roke Manor Research Ltd., January 2003.

[21] University of Southern California Information Sciences Institute. Internet Protocol. RFC 791, September 1981.

[22] Intel. Intel IXP2400 Hardware Reference Manual. Datasheet, Feburary 2004.

[23] Intel. IXP2400 Network Processor Reference Manual. Intel IXA Software Development Kit, May 2004.

[24] V. Jacobson, K. Nichols, and K. Poduri. An Expedited Forwarding PHB. RFC 2598, June 1999.

[25] H. Michael Ji and Michael Carchia. Fast IP Packet Classification with Configurable Processor. In *IEEE Global Telecommunications Conference*, pages 2268–2274, 2001.

[26] Wenyu Jiang, K. Koguchi, and H Schulzrinne. QoS evaluation of VoIP end-points. In *Proceedings of IEEE ICC*, pages 1917–1921, May 2003.

[27] S.S. Kanhere and H. Sethu. Low-Latency Guaranteed-rate Scheduling using Elastic Round Robin. In *Computer Communications 25*, pages 1315–1322, 2002.

[28] Brian Kantor and Phil Lapsley. Network News Transfer Protocol. RFC 977, February 1986.

[29] R. R. Kompella and G. Varghese. Reduced State Fair Queuing for Edge and Core Routers. In *NOSSDAV '04: Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video*, pages 100–105, 2004.

[30] Vijay P. Kumar, T. V. Lakshman, and Dimitrios Stiliadis. Beyond Best Effort: Router Architectures for the Differentiated Services of Tomorrow's Internet. *IEEE Communications Magazine*, May 1998.

[31] T.V. Lakshman and D. Stiliadis. High-Speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching. In *Proceedings of ACM SIGCOMM*, pages 203–214, 1998.

[32] L. Lenzini, E. Mingozzi, and G. Stea. Tradeoffs Between Low Complexity, Low Latency, and Fairness With Deficit Round-Robin Schedulers. *IEEE/ACM Transactions on Networking*, 12(4):681–693, August 2004.

[33] Fulu Li, Nabil Seddigh, Biswajit Nandy, and Diego Matute. An Empirical Study of Today's Internet Traffic for Differentiated Services IP QoS. In *ISCC*, April 2000.

[34] D. Lin and R. Morris. Dynamics of Random Early Detection. In *ACM SIGCOMM Computer Communication Review*, 1997.

[35] Ying-Dar Lin, Yi-Neng Lin, Shun-Chin Yang, and Yu-Sheng Lin. DiffServ over Network Processors: Implementation and Evaluation. In *High Performance Interconnects*, pages 121–126, August 2002.

[36] Z. Lotker and B. Patt-Shamir. Nearly Optimal FIFO Buffer Management for Diff-Serv. In *PODC: Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 134–143, 2002.

[37] Chairman Federal Communications Commision Michael K. Powell. Remarks. National Association of Regulatory Commissioners, March 2004.

[38] Nortel Networks. Introduction to Quality of Service (QoS). White Paper, January 2003.

[39] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. RFC 2474, December 1998.

[40] Andrew M. Odlyzko. Internet Traffic Growth: Sources and Implications. In *SPIE*, pages 1–15, 2003.

[41] M.H. Overmars and A.F. van der Stappen. *Journal of Algorithms*, volume 21, pages 629–656. November 1996.

[42] J. Postel and J. Reynolds. File Transfer Protocol. RFC 959, October 1985.

[43] Jonathan B. Postel. Simple Mail Transfer Protocol. RFC 821, August 1982.

[44] Radisys. ENP-2611 Hardware Reference. Technical Support Library, October 2004.

[45] J. Rexford, F. Bonomi, A. Greenberg, and A. Wong. A Scalable Architecture for Fair Leaky-Bucket Shaping. In *IEEE INFOCOM*, 1997.

[46] J. L. Rexford, A. G. Greenberg, and F. G. Bonomi. Hardware-Efficient Fair Queueing Architectures for High-Speed Networks. In *IEEE INFOCOM '96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation*, March 1996.

[47] Lawrence G. Roberts. The Evolution of Packet Switching. In *Proceedings of the IEEE*, pages 1307–1313, November 1978.

[48] Tyrell Sassen. Limitations of the Hierarchical Intelligent Cuttings Packet Classification Algorithm. In *Proceedings of SATNAC*, 2005.

[49] N. Shah. Understanding Network Processors. Master's thesis, Department of Electrical Engineering and Computer Science, Univ. of California, Berkeley, June 2001.

[50] Niraj Shah, William Plishker, and Kurt Keutzer. NP-Click: A Programming Model for the Intel IXP1200. In *2nd Workshop on Network Processors (NP-2) at the 9th International Symposium on High Performance Computer Architecture (HPCA-9)*, Anaheim, CA, February 2003.

[51] M. Shreedhar and G. Varghese. Efficient Fair Queuing using Deficit Round-Robin. In *IEEE/ACM Transactions on Networking*, 1996.

[52] Sumeet Singh, Florin Baboescu, George Varghese, and Jia Wang. Packet Classification Using Multidimensional Cutting. In *SIGCOMM*, pages 213–224, 2003.

[53] Tammo Spalink, Scott Karlin, Larry Peterson, and Yitzchak Gottlieb. Building a robust software-based router using network processors. In *Proceedings of the ACM SOSP*, pages 216–229, New York, NY, USA, 2001. ACM Press.

[54] E. Spitznagel, D. Taylor, and J. Turner. Packet Classification Using Extended TCAMs. In *Proceedings of IEEE International Conference on Network Protocols (ICNP)*, 2003.

[55] Edward W. Spitznagel. Compressed Data Structures for Recursive Flow Classification. Technical report, Department of Computer Science and Engineering, Washington University in St. Louis, May 2003.

[56] D. Srinivasan and W. Feng. Performance Analysis of Multi-dimensional Packet Classification on Programmable Network Processors. In *29th Annual IEEE Conference on Local Computer Networks*, pages 360–367, November 2004.

[57] Deepa Srinivasan. Performance Analysis of Packet Classification Algorithms on Network Processors. Masters Thesis, Oregon Health and Science University, May 2003.

[58] V. Srinivasan. A Packet Classification and Filter Management System. Technical report, Microsoft Research, 2001.

[59] V. Srinivasan, S. Suri, and G. Varghese. Packet Classification using Tuple Space Search. In *Proceedings of SIGCOMM*, pages 135–146, 1999.

[60] V. Srinivasan, S. Suri, G. Varghese, and M. Waldvogel. Fast and Scalable Layer four Switching. In *Proceedings of ACM Sigcomm*, pages 203–214, September 1998.

[61] Agere Systems. Advanced PayloadPlus® APP540 and APP520 Packet-Optimized Network Processors. Product Briefs, January 2003.

[62] Cisco Systems. Cisco Performance Routing Engine 2 for the Cisco uBR10012 Router. Data Sheet, May 2005.

[63] Y. Tang, L. Qian, B. Bou-Diab, A. Krishnamurthy, G. Damm, and Y. Wang. High-Performance Implementation for Graph-Based Packet Classification Algorithm on Network Processor. *IEEE International Conference on Communications*, 2:1268–1272, June 2004.

[64] David E. Taylor. *Models, Algorithms and Architectures for Scalable Packet Classification.* PhD thesis, Washington University, Sever Institute of Technology, Department of Computer Science and Engineering, August 2004.

[65] David E. Taylor and Jonathan S. Turner. Towards a Packet Classification Benchmark. Technical Report WUCS-03-42, Washington University Computer Science Department, May 2003.

[66] David E. Taylor and Jonathan S. Turner. ClassBench: A Packet Classification Benchmark. In *INFOCOM*, 2005.

[67] Verizon. Online. http://www.verizon.com/, 2005.

[68] Li Zheng, Liren Zhang, and Dong Xu. Characteristics of Network Delay and Delay Jitter and its Effect on Voice over IP (VoIP). In *Proceedings of IEEE ICC*, 2001.

Appendix A

The ClassBench Packet Classification Benchmark Suite

ClassBench was created as a tool for benchmarking packet classification architectures. Real-world rule-sets are not usually publicly available and therefore difficult to obtain. This tool seeks to alleviate this problem by using a statistical approach to generate synthetic rule-sets. The databases are created in such a way that they have properties resembling real rule-sets. The suite contains two tools: a filter set generator and a trace generator.

The *filter set generator* uses a number of options to generate the rule-sets. The *parameter file* is the most important option, this contains a number of low-level metrics regarding the statistics and distribution of the rule-set. There are a number of parameter files included in the ClassBench suite. These are based on the real rule-sets of 12 common packet classification applications, such as firewalls, VPNs and ACLs. The other parameters are used to adjust the high-level properties of the generation process. The *size* parameter specifies the number of rules to generate, this is known as the requested number of rules. Once this number of rules has been generated, the rule-set is pruned to remove redundant rules, this reduces the final size of the rule-set slightly. The *smoothing* parameter allows the user to generate larger rule-sets by introducing additional address aggregates. The *scope* parameter specifies if the generator favours larger or smaller rule bit-prefixes for the output rule-set.

The *trace generator* is used to generate test-data for a particular rule-set. This tool takes an earlier generated rule-set and a number of tuning parameters as inputs and produces a specified number of packet headers as an output. All packets in the test data will match

at least one rule in the generated rule-set. The *scale* parameter will specify the number of packets to generate and the *locality* parameter will specify the data locality of the test data.

A.1 Data Generated for the Evaluation Framework

The above mentioned tools were used to generate all rule-sets and test packets for the evaluation framework. Table A.1 shows the parameters used for the filter set generator, these were kept constant and only the size parameter was changed to create the various sized rule-sets. The parameter file that was used was modelled on the rule-set of a VPN, this was the closest representation of the type of rules that would be found in the rule-set for a DS-enabled router. The default values were used for the other 2 parameters.

Parameter	Value
parameter file	acl1_seed
size	32 – 16384
smoothing	30
scope	-0.5

Table A.1: The parameters used to generate the rule-sets.

To generate the test data, the parameters in Table A.2 were used. The size of the trace data was specified to be 10x the size of the rule-set. This ensured that each rule in the database had approximately the same number of matching packets. The locality parameter was left at the default value. Since none of the algorithms in this study uses any type of caching, this parameter will have no effect on the performance results.

Parameter	Value
scale	10
locality	1

Table A.2: The parameters used to generate the test packets.

A.2 Custom Tools

Several custom made tools were used to convert the data generated by the ClassBench suite into a format that could be used in the evaluation framework.

The *GenerateDB* tool took the generated rule-sets and converted them into C include rules. These were then included in the preprocessing stages of the various algorithms and used to generate the algorithm data-structures.

The *GenerateStream* tool used the headers generated by the benchmark suite to create minimum-sized, Ethernet test-packets that could be imported into the IXP2400 simulator. The tool generated TCP/IP packets with the headers specified by the Trace Generator and no body.

Appendix B

The Intel IXP2400 Network Processor

This research uses the Intel IXP2400 Network Processor (NP). This is a member of Intel's 2nd-generation of NPs and is designed for access and edge-network applications. Figure B.1 shows the functional blocks of the NP, the ones that are applicable to this research are described in more detail below.

B.1 XScale Host Processor

The host processor is responsible for all the higher-level operations in the NP. These will include control and management functions, as well as higher-level packet processing. The XScale processor will run a real-time operating system. This research uses Montavista Linux 3.0, however it would also be possible to use VxWorks. For packet classification

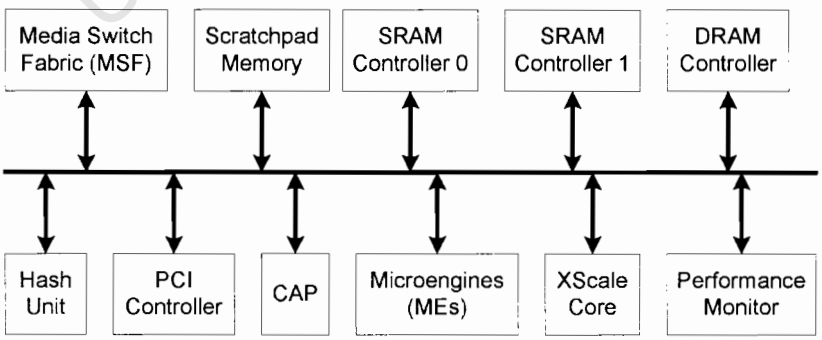

Figure B.1: The functional units of the Intel IXP2400

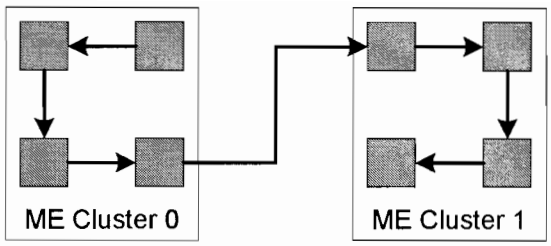

Figure B.2: The adjacent MEs of the Intel IXP2400

the host processor would most likely execute the preprocessing stages of the algorithm to generate the necessary data-structures.

B.2 The Microengines

The microengines (MEs) are the main network processing units of the NP. The IXP2400 has 8 32-bit MEs, which are divided into 2 clusters. The instruction set of the MEs is tailored for packet processing applications. Besides the more common ones, there are instructions to find the first set bit in a register (FFS), access the CAM unit (CAM_LOOKUP) and media and switch fabric (MSF) and support for hardware queues in memory (SCRATCH). Each ME has a clock speed of 400 or 600 MHz and can have up to 8 threads with no context-switching overhead. The threads increase the parallelism of the design and help to hide the latency of the external memory. There a number of different register kinds in each ME: 256 general purpose registers, 512 transfer registers – used to transfer data to and from external memory – and 128 next-neighbour (NN) registers. The NN registers are used to transfer data between adjacent MEs. The adjacent MEs are shown in Figure B.2, data can only be transfered as indicated by the arrows between the MEs.

B.3 SRAM

The SRAM controller of the IXP2400 has two independent channels, which each support 64 MB of memory. The IXP2400 does not contain any internal SRAM but this is added externally and interfaced through the controller. This memory is usually used to store the larger data-structures used in packet processing, like the packet classification structures

used in this research. The controller allows a number of memory accesses to be queued and MEs and XScale processor to be signalled once the operations are complete.

B.4 DRAM

The DRAM controller functions in a similar manner to the SRAM controller, however there is only one channel with support for a maximum of 2 GB of memory. This memory is often used to store packet bodies.

B.5 Media and Switching Fabric (MSF)

The media and switching fabric (MSF) connects the IXP2400 to the physical network interfaces. This unit supports a number of different framing and MAC protocols, namely UTOPIA 1/2/3, POS-2, SPI-3 and CSIX.

B.6 Scratchpad Memory

The scratchpad memory is a 16 KB block of internal memory. It is faster than both the SRAM and DRAM and supports both ring and atomic operations. This is generally used as a buffer for storing packet information.

B.7 Hash Unit

The hash unit is a co-processor that can be used by the MEs and XScale processor to perform hash calculations. The size of the calculation and the hash equations can be configured by setting a few registers.

Appendix C

The Intel IXP2400 Software Development Kit

The IXP2400 simulator forms part of the IXP software development kit and provides a cycle-accurate simulator of the NP. This is attached to the debugging system of the development environment and allows code to be tested and profiled before being executed on a physical IXP2400.

The benefit of this system is that it is possible to gather information about the inner workings of the processor during execution. This can allow one to read and change the values stored in the registers and memory of the system.

C.1 Packet Classification ME Initialisation

The simulator also contains a scripting language that can be used to set the parameters of the NP. This is used for the initialisation of the MEs and memory, which is usually done by the XScale processor. The standard initialisation functions that are provided with the development kit were used in the evaluation framework. The scripting engine was also used to create the packet classification data-structures in memory.

Since the development environment does not provide a simulation for the XScale processor, the preprocessing steps for the architecture were executed on a standard desktop computer. The code for these functions was written in C, using the standard libc libraries, and would only need to be recompiled to be compatible with the XScale processor.

The rule-sets for the evaluation framework are included in the preprocessing stages as described in Appendix A. These were then used to calculate the data-structures required by the various algorithms, which were then written to a script file to be imported by the simulator during startup.

The script file used the SET_SRAM instruction to specify the data to be written to each memory location. The starting addresses and size of the various algorithms were also calculated in the preprocessing stage and written to a file that was included in the source code for each algorithm. This meant that the packet classification algorithms needed to be recompiled each time the rule-set was changed.

C.2 Packet Simulation

The network ports of the IXP2400 are also simulated by the development environment. This allows the simulator to create virtual traffic on the four ports of the NP, which can then be processed by the MEs. The traffic streams for the evaluation framework were imported into the simulator from the ClassBench suite. Details regarding this can be found in Appendix A. The simulator allows the user to specify the transmission rates and traffic streams for each node. Statistics regarding the actual data rates and queue utilisation are recorded by the simulator and made available for export. These were the metrics used for the data throughput analysis in Section 5.3.