DiffServ Edge Routers over Network Processors: Implementation and Evaluation

Ying-Dar Lin and Yi-Neng Lin, National Chiao Tung University
Shun-Chin Yang and Yu-Sheng Lin, Industrial Technology Research Institute

Abstract

Network processors are emerging as a programmable alternative to the traditional ASIC-based solutions in scaling up the data plane processing of network services. This work, rather than proposing new algorithms, illustrates the process of, and examines the performance issues in, prototyping a DiffServ edge router with IXP1200. The external benchmarks reveal that although the system can scale to wire speed of 1.8 Gb/s in simple IP forwarding, the throughput declines to 180–290 Mb/s when DiffServ is performed due to the double bottlenecks of SRAM and microengines. Through internal benchmarks, the performance bottleneck was found to be able to shift from one place to another given different network services and algorithms. Most of the results reported here should be applicable to other NPs since they have similar architectures and components.

Increasing link bandwidth demands faster nodal processing, especially of data plane traffic. Nodal data plane processing ranges from routing table lookup to various classifications for firewalls, differentiated services (DiffServ), and Web switching. The traditional general-purpose processor architecture is no longer sufficiently scalable for wire speed processing, and some application-specific integrated circuit (ASIC) components or co-processors are commonly used to *offload* the data plane processing, while leaving only control plane processing to the original processor.

Several ASIC-driven products have been announced in the market, such as acceleration cards for encryption/decryption, virtual private network (VPN) gateways, layer 3 switches, DiffServ routers, and Web switches. While accelerating data plane packet processing with special hardware blocks, much wider memory buses, and faster execution processes, these ASICs lack the flexibility of reprogrammability and have a long development cycle, usually of months or even years. The cost of possible design failures is also high.

Network processors are emerging as an alternative solution to ASICs for providing reprogrammability while retaining scalability for data plane packet processing. This study employed the Intel IXP1200 [1] network processor, which consists of one StrongARM core and six co-processors, referred to as *microengines*, so that developers can embed the control plane and data plane traffic management modules into the StrongARM core and microengines, respectively. Scalability concerns in data plane packet processing could be satisfied with the four zero context switching overhead hardware contexts in each of the six microengines and the instructions specifically for networking.

This work was supported in part by a research grant from Industrial Technology Research Institute (ITRI), Taiwan, and in part by a research grant and an equipment donation from Intel Corporation.

Spalink, Karlin, Peterson, and Gottlieb [2] demonstrated and evaluated the IXP1200 in IP forwarding, concluding that the SDRAM storing packets is the bottleneck. However, such results cannot be generalized to today's complex services, which may need a great deal of SRAM table accesses and computing power. This work therefore aims to implement a more sophisticated service, DiffServ, using two existing algorithms for classification and scheduling, and identify scalability issues and possible performance bottlenecks in IXP1200. Two topics in benchmarking the implemented system are investigated. First, can this DiffServ implementation scale to a large number of classification rules? Although DiffServ defines only a limited number of traffic classes, the number of classification rules in the DiffServ edge routers (i.e., the number of flows) could be large. Second, where are the potential *bottlenecks* and what are their causes? The exact bottleneck is anticipated to depend on the specific service and its *algorithmic* implementation.

The rest of this article is organized as follows. We briefly review the architecture of IXP1200. We then present the design and implementation of DiffServ over IXP1200. Next, we illustrate the results of external and internal benchmarks through experiment and simulation. A summary of this work and possible ways to remove bottlenecks are discussed.

Architecture of IXP1200

Closely examining the hardware architecture of IXP1200 shown in Fig. 1 helps to elucidate our DiffServ implementation. The 32-bit 200 MHz StrongARM core governs the initialization of the whole system and part of the packet processing. A memory management unit is also included to translate virtual addresses into physical addresses and control memory access permission.

The six 200 MHz microengines, supporting four hardware contexts (i.e., *threads*), are primarily used to receive, manipulate, and transmit packets. For networking purposes, microengines also support zero context switching overhead, single-cycle ALU with shifter, and other specifically designed instructions for bit, byte, and longword operations.

The SRAM is used to store lookup *tables* and *pointers* in scheduling queues for packet forwarding, while the SDRAM is used to store mass data of *packets*. The 64-bit IX bus interface unit is responsible for servicing medium access control (MAC) interface ports on the IX bus, and moving data to and from the receive and transmit first-in first-out buffers (FIFOs). It provides a 4.2 Gb/s (64 bits at 66 MHz) interface to MAC devices, meaning that it can afford 2.1 Gb/s of the input ports and 2.1 Gb/s of the output ports. In addition, two IXP1200 network processors can be directly supported on the IX bus without additional support logic.

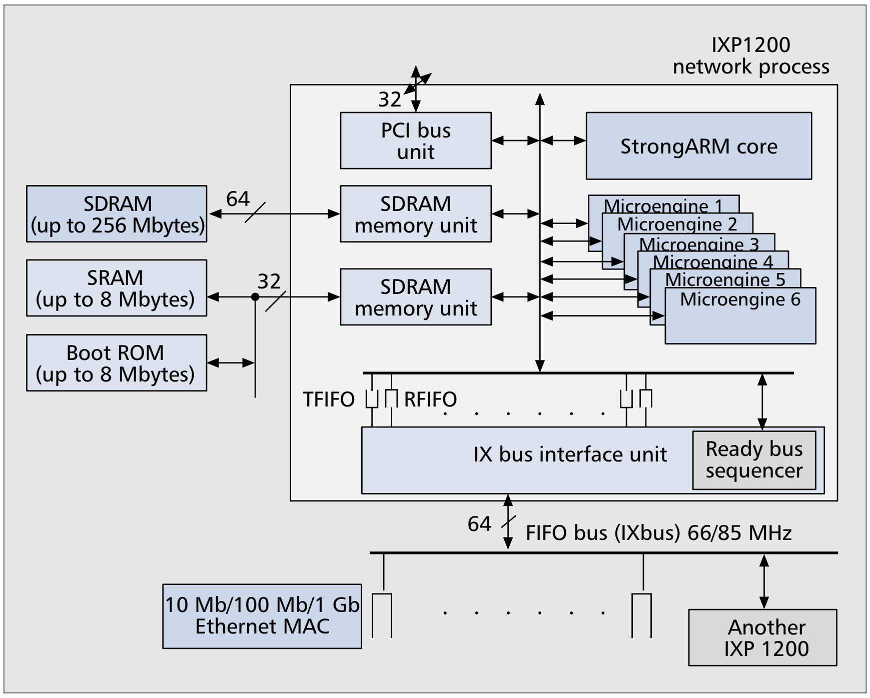


Figure 1. *The hardware architecture of IXP1200.*

The operations of IXP1200 hardware components when handling packet forwarding services are described below. At boot time, the StrongARM loads the boot image from boot ROM or via serial/Ethernet connection, and initializes other functional units, including loading the routing table into SRAM and microcode into microengines. The system is now ready to receive packets. When the ready bus sequencer detects an incoming packet in a MAC, it notifies the corresponding *receiver* thread to retrieve and store the packet in the receive FIFO (RFIFO). After completing the routing table lookup, the receiver thread moves the packet to SDRAM in order to wait to be forwarded. A transmitter thread of another microengine later forwards the packet in SDRAM through the transmit FIFO (TFIFO) to another MAC. Multiple receiver, transmitter, and scheduler threads may be distributed to six microengines, although some restrictions apply.

Design and Implementation of DiffServ on IXP1200

This section briefly introduces DiffServ and then explains how to map DiffServ components onto an IXP1200. The implementation of two major components, classifier and scheduler, in DiffServ using two algorithms, Multidimensional Range Matching and the weighted form of Deficit Round-Robin, is described.

DiffServ Briefing

DiffServ [3] mechanisms enable users to receive different levels of service from a provider to support various types of applications. According to the service configuration in a DiffServ edge node, packets are classified with multiple fields (MFs), leaky bucket policed, and marked to receive a particular forwarding per-hop behavior (PHB), which defines how packets with a particular behavior are treated at this node. Each predefined PHB is mapped to one DiffServ code point (DSCP) value used in class-based scheduling: expedited forwarding (EF) or one of four assured forwardings (AFs).

The service differentiation of packets is often manifested as delay and loss rate. Packets of higher classes are more likely to be scheduled before those of lower classes, resulting in lower latency and loss rate.

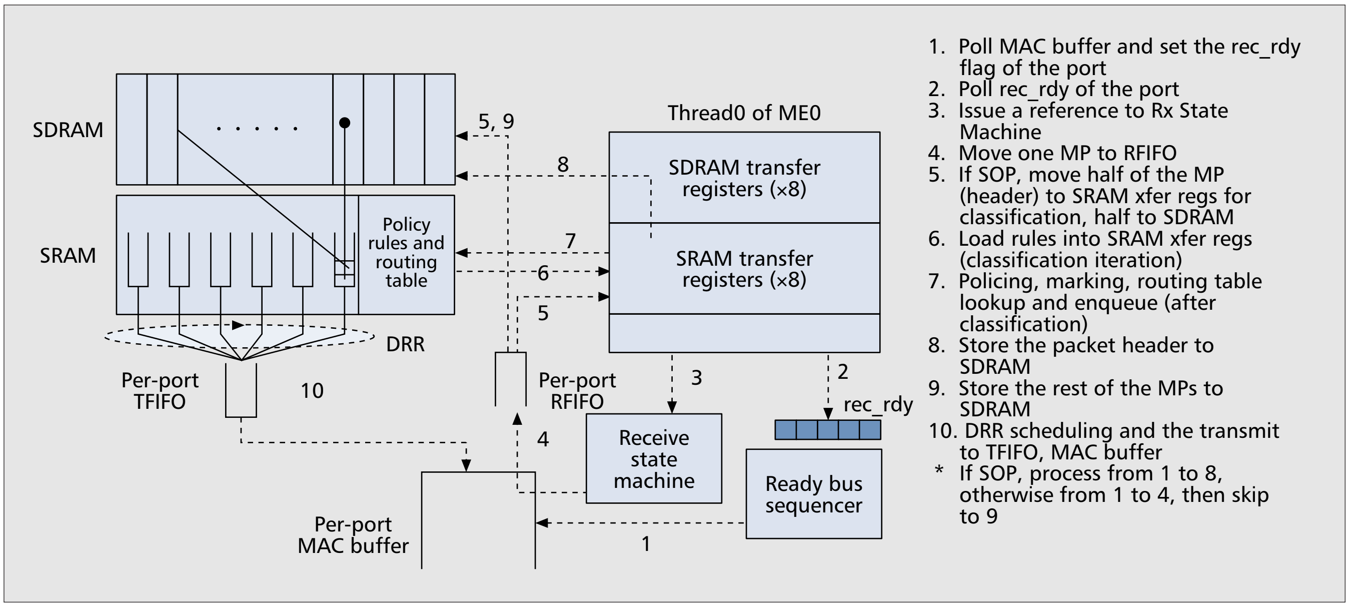
Detailed Packet Flow in IXP1200

Figure 2 illustrates the key components and the packet processing flow. The ready bus sequencer periodically polls the MAC buffer and sets the receive flag in a global rec_rdy register when a packet comes. Once the receiver thread responsible for the MAC port detects the flag, it asks the receive state machine to move the packet, in units of 64-byte MAC packets (MPs), a basic data unit in the system, from the MAC buffer into RFIFO.

Mapping DiffServ Components

Figure 3 shows the software architecture of DiffServ and its corresponding task allocation on IXP1200. Six modules (the shaded blocks) are inserted into the original software of simple IP forwarding.

The DiffServ processing is described below. Once received at a transfer register from an RFIFO and verified as legal, a packet header is passed to the range matching classifier for the matching process. If the packet's header matches one of the classification rules and is classified as, for example, EF traffic, it is admitted or discarded according to the policing bandwidth specified in the classification rule. If admitted, it is marked with a DSCP in the header. After longest prefix matching in routing table lookup, the packet is queued in the corresponding queue of the output port, and waits to be scheduled; that is, the packet's descriptor is enqueued in SRAM while the packet itself is stored in SDRAM. The scheduler thread chooses one transmitter thread and assigns it a port, which contains six queues (one EF, four AFs, and one best effort, BE), to serve. The transmitter thread examines the queue with the highest priority to determine whether a packet is waiting to be sent, and whether the queue has sufficient quanta as credits for transmitting that packet in Deficit Round-Robin scheduling. If it has enough quanta, the trans-



SDRAM
SRAM
Policy rules and routing table
DRR
Per-port TFIFO
10
5, 9
8
7
6
5
Thread0 of ME0
SDRAM transfer registers (×8)
SRAM transfer registers (×8)
3
2
rec_rdy
Per-port RFIFO
4
Receive state machine
Ready bus sequencer
Per-port MAC buffer
1
1. Poll MAC buffer and set the rec_rdy flag of the port
2. Poll rec_rdy of the port
3. Issue a reference to Rx State Machine
4. Move one MP to RFIFO
5. If SOP, move half of the MP (header) to SRAM xfer regs for classification, half to SDRAM
6. Load rules into SRAM xfer regs (classification iteration)
7. Policing, marking, routing table lookup and enqueue (after classification)
8. Store the packet header to SDRAM
9. Store the rest of the MPs to SDRAM
10. DRR scheduling and the transmit to TFIFO, MAC buffer
* If SOP, process from 1 to 8, otherwise from 1 to 4, then skip to 9

■ Figure 2. *Detailed DiffServ packet flow in IXP1200.*

mitter thread fetches the packet's descriptor in SRAM and sends the entire packet in SDRAM to TFIFO for output. Otherwise, the thread examines the next queue for packets to be sent, and the quanta.

The 24 threads are equally divided into two groups: eight Fast Ethernet (FE) 10/100M ports and one Gigabit Ethernet GbE) port. Each group has 12 threads, eight of which are used as receivers (assigned to two microengines), three as transmitters, and one as a scheduler (assigned to one microengine). Each 10/100M receiver thread is responsible for a specific 10/100M port, while eight GbE receiver threads serve one GbE port. The transmitter threads, however, are not bound to specific ports. They output packets to ports according to assignments from the scheduler thread. Static task allocation, instead of dynamic task allocation, is employed for the following reasons. First, the 1000 control store of a microengine may not be sufficiently large to hold microcode of two threads of different types, for example, receiver (1012 instructions) and transmitter (552 instructions), whose summed size of instructions exceeds the control store size. However, the transmitter and scheduler (144 instructions), whose summed size is below 1024, can coexist in one microengine. Therefore, threads of the same type are best grouped in one microengine. Second, choosing dynamic allocation complicates the programming, and the communication overhead between threads or microengines would be huge as tasks could not be clearly divided among threads.

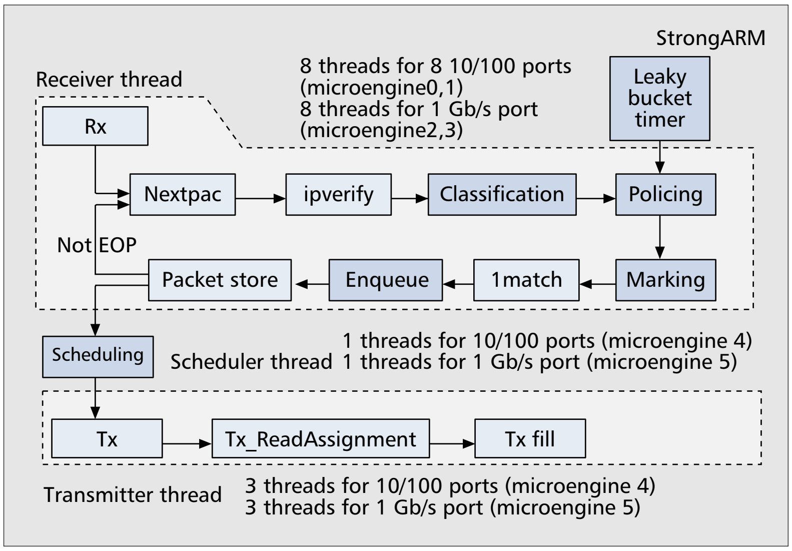
Algorithm Adoption and Implementation

Related Work — Classification and scheduling are two critical modules that influence the performance of a DiffServ implementation. Several methods have been proposed for the above two purposes; however, many of them are not practically applicable due to limitations of the platform, including memory size (2 Mbytes of SRAM) and coding overhead; for example, Recursive Flow Classification [4], which has an unstable memory size requirement ranging from 1 to 1000 Mbytes; similar behavior can also be seen in Cross-Producting [5].

Another example is the Grid-of-Tries [5], which is also a classification algorithm and has a lower memory requirement. Nonetheless, the complexity of the algorithm makes implementation with microcode difficult. Several scheduling algorithms are not considered for the same reason, including, for example, Weighted Fair Queuing [6], which involves complex multiplications.

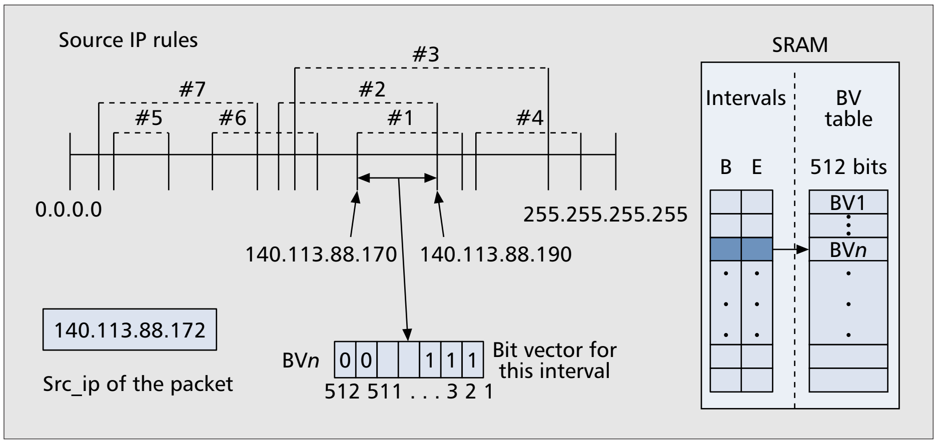
Multidimensional Range Matching [7] is thus used as a classifier, to exploit its more stable and lower memory requirement, and efficiency in setting up flexible classification rules. The weighted form of Deficit Round-Robin [8] is adopted in the scheduler, which can be implemented easily (requiring only *addition*) and effectively ensures weighted sharing among various flows. The following two subsections briefly describe the implementation of these two algorithms.

Classifier — The concept of Multidimensional Range Matching used to implement the classifier is described below. The n classification rules in a dimension form at most $(2n - 1)$ intervals, and at most n rules may overlap in one interval. Each interval is associated with a bit vector (BV), which is 512 bits in this implementation and stored in SRAM to rep-



StrongARM
Receiver thread
8 threads for 8 10/100 ports (microengine0,1)
8 threads for 1 Gb/s port (microengine2,3)
Leaky bucket timer
Rx
Nextpac
ipverify
Classification
Policing
Not EOP
Packet store
Enqueue
1match
Marking
Scheduling
Scheduler thread
1 threads for 10/100 ports (microengine 4)
1 threads for 1 Gb/s port (microengine 5)
Tx
Tx_ReadAssignment
Tx fill
Transmitter thread
3 threads for 10/100 ports (microengine 4)
3 threads for 1 Gb/s port (microengine 5)

■ Figure 3. *Data plane architecture of a DiffServ edge router over IXP1200.*



■ Figure 4. *Example and related tables for lookup in the source IP dimension.*

resent the rules overlapped in this interval. The space complexity is $O(n^2)$ because of the n classification rules and $(2n - 1)$ intervals.

Figure 4 presents an example of the matching process in the source IP dimension. When a packet arrives, the classifier performs a binary search in the interval table of each dimension with the corresponding files of the packet. When an interval is found for a dimension, the classifier retrieves the corresponding BV in the BV table. The classifier then ANDs the BVs from all intervals, and the index of the first nonzero bit in the result vector becomes the index of the first matched classification rule.

After the classifier returns the index of the matched classification rule, the policer and marker use the information contained in the rule in further processing. Each rule is associated with two additional fields, *last_arrival_time* and *token*, which are used to maintain per-flow leaky bucket. A timer is implemented by StrongARM to determine timing information. The *last_arrival_time* is the arrival time of the previous packet, and the *token* represents the number of quanta left after processing the last packet. The token field is increased with the product of a configured quanta rate, and the time interval between last and current arrivals. The total tokens available to the incoming packet can thus be determined, and a decision regarding its admission can be made.

Scheduler — The quantum size of each class can be set arbitrarily. Here, we set the ratio of the quantum between two adjacent classes in this system to two for simplicity. A packet is represented by a queue descriptor in SRAM. Each queue descriptor contains the count of MPs and the pointer to a link list of buffer descriptors, which point to the MPs of the packet stored in SDRAM. Once a packet is scheduled for transfer, the transmitter thread uses the addresses of the buffer descriptors and the buffer handle in the last buffer descriptor to locate all MPs. The former are used to map the start addresses of the MPs (buf_des_addr*64), and the latter is used to determine the number of valid bytes at the end of packet (EOP).

External and Internal Benchmarks

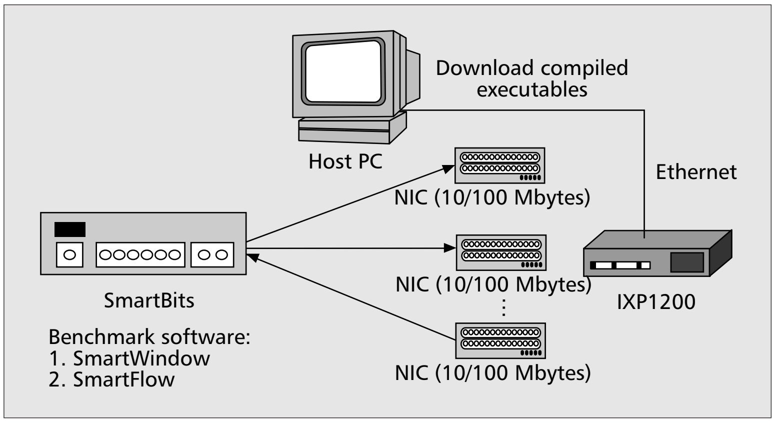
The performance of DiffServ has been evaluated in a number of studies [9–12]. However, most of these involve only simulations. Accordingly, this section considers two kinds of experiments: external and internal benchmarks. For the former, the functionality of DiffServ is validated, and the aggregated throughput achievable by the system while conforming to the PHBs is examined to determine scalability. Two other implementations of DiffServ, a Linear Search classifier over IXP1200 and a Range Matching classifier over a Pentium III 800 CPU containing 128 Mbytes SDRAM and running the Linux operating system, are also included for comparison with a Range Matching classifier over IXP1200. As shown in Fig. 5, the benchmark environment consists of three components: IXP1200, a host PC, and SmartBits; a host PC is used to remotely control the initialization and activities of IXP1200, while SmartBits is used to generate test patterns.

The internal benchmark involves software simulations of two DiffServ implementations on IXP1200 with classifiers implemented based on Linear Search and Range Matching, respectively. The aim is to observe the utilization of internal resources, as well as the performance bottlenecks. The simulations are conducted under WorkBench, which is a simulator of IXP1200. In this section we simulate eight 10/100M ports with three microengines, which have 12 threads allocated as described earlier.

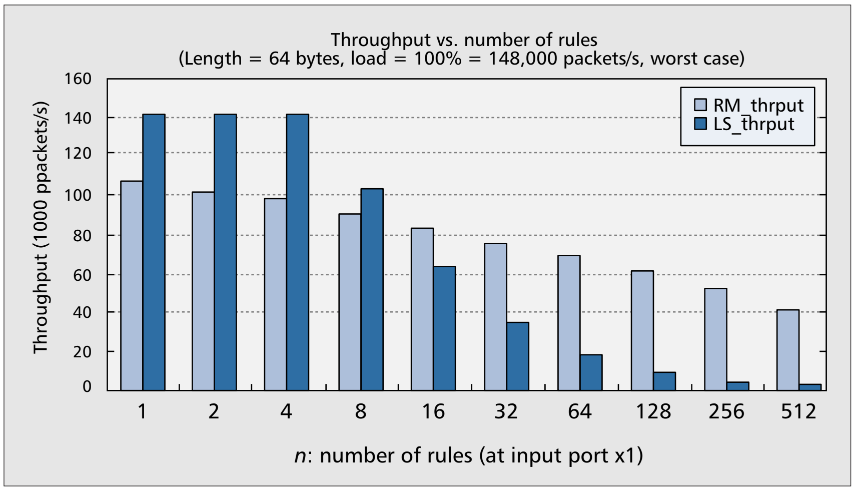
The Functionality Test

Figure 6 depicts the throughput of two DiffServ implementations for one input port. Although the time complexity of Range Matching is $O(n)$, the benchmark result in Fig. 6 shows a $k^*(\log n)$ decrease in the throughput as the number of classification rules increases. This is because when the number of classification rules is small (as in our experiment), the coefficient k, which represents the effect from binary searches of multiple dimensions, dominates the classification process. As for Linear Search, we can see that the throughput is linearly decreased as the number of classification rules increases.

Figure 7 depicts the throughput of four receivers that are receiving one EF and three AF flows from four input ports. The traffic of AF3 begins to be dropped at a load of 25 percent because the output link is fully utilized so that the packets of low priority are more likely to be dropped. The other three flows continue to consume the bandwidth until the output queue of AF2 is full due to the higher consumption of the link by the other two flows. Finally, all flows enter their steady state when EF flow reaches its bandwidth limit specified in the classification rule. The three AF flows obey the 2:1 traffic



■ Figure 5. *The benchmark environment.*



■ Figure 6. *Throughput of two DiffServ implementations with varying number of classification rules.*

proportion as designed earlier; the EF flow does not because its queue is not full.

In the latency test in Fig. 8 corresponding to Fig. 7, we observe that the EF flow has a very low latency under all load conditions. Before the load of 25 percent, every flow has the same latency because the queues are not full. We also observe that the latency of AF flows still obeys the 2:1 proportion, which means the delay in output queues dominates the total system delay.

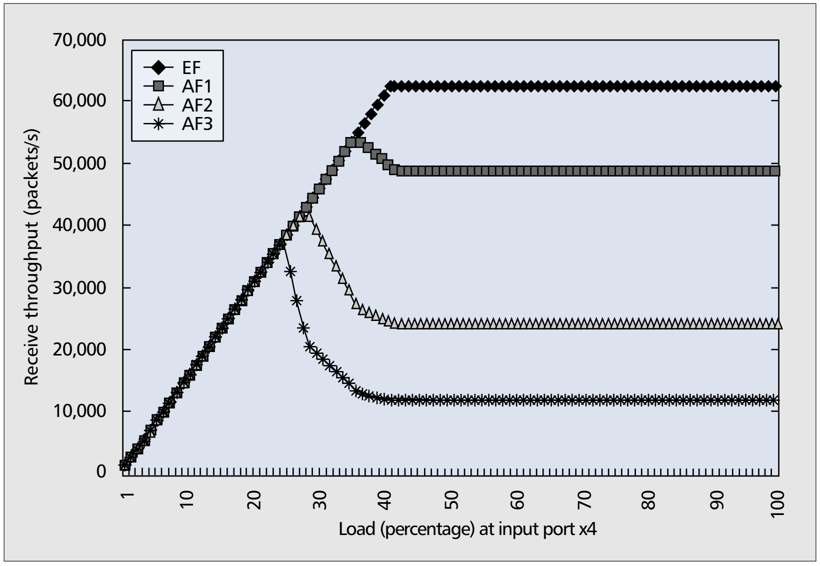
Scalability Test

Traffic Load — The methodology for testing fairness between flows within one input port is described below. The maximum load, which is 58 Mb/s, is first measured for a flow that yields no packet loss. The fairness among flows can then be examined with aggregated input load below and above 58Mb/s.

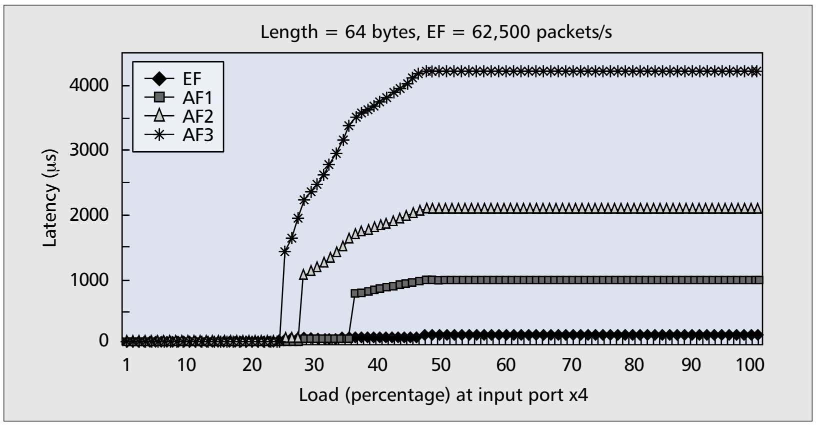
Figure 9 presents the throughputs of 500 flows under two load conditions, with each flow exactly matching the corresponding rule within 500 classification rules. The flows strictly follow their bandwidth settings when the input load is 50 percent of the wire speed 64-byte Fast Ethernet traffic, which is 74,400 ackets/s, and become unstable when overloaded. However, most of the flows are limited to their bandwidth settings at 148.8 packets/s or 74,400/500 packets/s.

Number of Microengines — Figure 10 presents the throughput of the receiver threads of different configurations. Naturally, the throughput of two threads in two microengines is around double that of a single thread. However, the throughput of four threads in a microengine is not four times that of one thread due to a lack of computing power. Furthermore, the throughput of eight threads is not double that of four threads, because of memory contention. Besides, the aggregated system throughput ranges from 180 Mb/s to 290 Mb/s, according to the number of classification rules, while the throughput of IP forwarding, which does the work of unshaded blocks in Fig. 3, is at wire speed.

Figure 10 yields some interesting observations concerning the bottlenecks of DiffServ for various input traffic allocations. A single port receiver thread can obtain sufficient computing power because the other three threads do not process packets, just poll the flag register; but the memory access takes so long that the thread cannot finish the processing of a packet in time to receive later packets. In the test of four input ports, whose corresponding threads are all in the same microengine, the bottleneck becomes the microengine because the four threads share the computing power of the microengine to perform complex computations in the Range Matching classification. The bottleneck remains the same in the test of eight input ports for the same reason. However, in the test of the whole system throughput, the bottleneck is again the SRAM, since the aggregated throughput is not the sum of the throughputs of eight 100 Mb/s ports and one gigabit port, although the computing power is doubled. The SRAM and microengines are called *double bottlenecks*, because the system can still suffer from one bottleneck after the other is solved.



■ Figure 7. *Priority and bandwidth control test (length = 64 bytes, EF = 62,500 packets/s).*



■ Figure 8. *4 (EF, AF1–3) to 1 latency test.*

	Active (percentage)	Rate
Microengine 0 (recv)	61.7	123.4 MIPS
Microengine 1 (recv)	72.2	144.5 MIPS
Microengine 2 (sche, xfer)	68.5	137.0 MIPS
SDRAM	9.3	594.8 Mb/s
SRAM	55.1	1764.1 Mb/s

■ Table 1. *Component statistics of the system with a Linear Search classifier.*

	Active (percentage)	Rate
Microengine 0 (recv)	99.7	199.4 MIPS
Microengine 1 (recv)	99.8	199.6 MIPS
Microengine 2 (sche, xfer)	96	192 MIPS
SDRAM	13	831.4 Mb/s
SRAM	35.3	1130.2 Mb/s

■ Table 2. *Component statistics of the system with a Range Matching classifier.*

Not shown in Fig. 10 is the throughput, 20.5 Mb/s, of the Linux-based Range Matching DiffServ when the number of classification rules is 512. The throughput is almost the same as that of one thread in IXP1200, implying that IXP1200 outperforms the general PC with its multithreaded processing power. The low performance of a higher-clock-rate general PC is due to operating system overhead and memory architecture. Without the need to interrupt the operating system from a user space NIC driver when a packet arrives, IXP1200 directly moves the packet into either the SRAM transfer registers for computation or SDRAM for temporary storage. The transfer is performed by the receive state machine without the involvement of microengines. The longer time required to access SDRAM in a PC than SRAM in IXP1200 also greatly impacts lookups of the classification rules and routing table.

Simulation: Linear Search

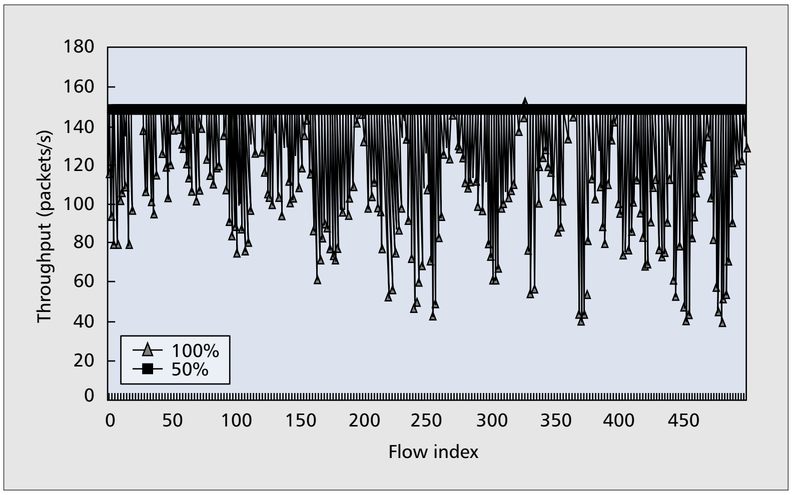
Since the most time- and computing-power-consuming phase in the system is the classification phase due to its rapid memory accesses and mass computations, we are interested in the effects of classifiers using different algorithms. In this section we simulate the system in which the classifier is implemented using Linear Search, and obtain the utilization and throughput of some hardware components such as MEs, SRAM, and SDRAM.

As Table 1 shows, the simulation of the Linear Search classifier yields two observations. The first is the low utilization of SDRAM. This is because packet forwarding, the main consumer of SDRAM, is not critical in DiffServ.

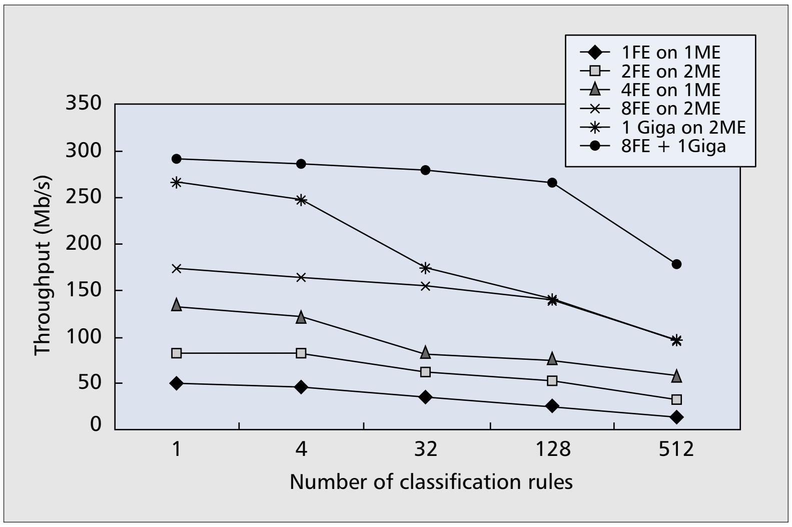
Second, both receiver microengines and SRAM are not fully utilized, but 70 and 55 percent utilized, respectively, while the actual throughput of the system is low. This can be explained as follows. Although the utilization of SRAM is only 55 percent, it is a bottleneck because SRAM access from receiver threads performing Linear Search is bursty, meaning that the bandwidth of SRAM is not used until bursty access from receiver threads. Moreover, sometimes all the threads in a microengine wait for SRAM access and thus cause the microengine to be idle.

Simulation: Range Matching

In this section we simulate the system in which the classifier is implemented using Range Matching. The utilizations of SDRAM and SRAM, as Table 2 shows, are again low at 13 and 35.3 percent, respectively. The former can be explained similarly as in the previous section. The lower SRAM utilization than with Linear Search classifier, on the other hand, illustrates the effects of the reduced frequency of SRAM accesses as a benefit of a more complex Range Matching data structure. However, the receiver microengines are almost fully utilized. This is due to complex computation in the Range Matching including the binary search for intervals in each dimension, logic operations on the resulting BVs, first active bit indexing, and



■ Figure 9. *Flow fairness test (Len = 64 bytes, 500 flows, BW/flow = 74,400/500 = 148 packets/s, average case).*



■ Figure 10. *Aggregated throughput (length = 64 bytes, worst case).*

rule retrieving. Computing power can therefore be identified as a performance bottleneck in the Range Matching DiffServ.

Summary

This study introduces the architecture and packet flow of DiffServ implementation in IXP1200. Two key modules in DiffServ, the classifier and scheduler, that adopt Multidimensional Range Matching and Deficit Round-Robin, respectively, are implemented in IXP1200 microcodes. Finally, external and internal benchmarks were applied to determine the bottlenecks in the implementation. Most of the result reported here should be applicable to other NPs since they have similar architectures and components.

The external benchmarks, which present the performance figures under different combinations of number of rules or flows, traffic load, and number of microengines, have illustrated that the implementation can well support PHBs in DiffServ at an aggregated throughput of 282 Mb/s (550kpps), as shown in Fig. 10. Both external and internal benchmarks identify the double bottlenecks of both SRAM and microengines in the Range Matching DiffServ— the Range Matching DiffServ could still suffer from one bottleneck after the other is solved. Although the SDRAM is the bottleneck in IP forwarding, the bottleneck may shift from one functional unit to another, depending on the specific service, algorithm, and the way input traffic is allocated to threads. Moreover, the SRAM bottleneck is found to occur not necessarily at 100 percent utilization, but even at 55 percent when the access is bursty.

Four methods are presented to solve the bottleneck of SRAM access that results in low utilization of receiver microengines. First, the routing table may be stored in SDRAM in the hope of offloading SRAM. Second, one large SRAM may be divided into many smaller banks at different interfaces, reducing the queuing delay of requests in the command queue, if the requested addresses are in different memory banks. Even some redundant memory modules may also be used, possibly with an access arbitrator, to store many copies of the routing table and classification rules to enhance accessibility. Third, a new memory architecture, such as quad data rate SRAM with a peak bandwidth of up to 1.6 Gbytes/s/channel (two to three times that supported by SRAM), may be adopted. However, a new interface between the memory and other functional units may be required. Finally, an additional cache (or *content addressable memory*) can be used to reduce the number of times memory is accessed, because traffic in the same time period normally shows locality in lookups of classification rules and routing tables.

Acknowledgment

The authors like to gratefully thank the anonymous reviewers who provided detailed revision suggestions on an earlier draft of this article.

Reference

[1] IXP1200 Data Sheet, Intel doc. no. 278298-004, May 2000.
[2] T. Spalink *et al.*, "Building a Robust Software-Based Router Using Network Processors," *Proc. 18th ACM Symp. Op. Sys. Principles.*
[3] S. Blake *et al.*, "An Architecture for Differentiated Services," IETF RFC 2475, Dec. 1998.
[4] P. Gupta, and N. McKeown, "Packet Classification on Multiple Fields," *ACM SIGCOMM '99.*
[5] V. Srinivasan *et al.*, "Fast and Scalable Layer Four Switching," *ACM SIGCOMM '98.*
[6] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queuing Algorithm," *ACM SIGCOMM '89.*
[7] T. V. Lakshman, and D. Stiliadis, "High-Speed Policy-based Packet Forwarding Using Efficient Multidimensional Range Matching," *ACM SIGCOMM '98.*
[8] M. Shreedhar and G. Varghese, "Efficient Fair Queuing Using Deficit Round-Robin," *IEEE/ACM Trans. Net.*, June 1996, vol. 4, no. 3, pp. 375–85.
[9] L. V. Nguyen, T. Eyers, and J. F. Chicharo, "Differentiated Service Performance Analysis," *5th IEEE Symp. Comp. and Commun.*, 2000, pp. 328–33.
[10] J. K. Muppala, T. Bancherdvanich, and A. Tyagi, "VoIP Performance on Differentiated Services Enabled Network," *IEEE Int'l. Conf. Net.*, 2000, pp. 419–23.
[11] J. Harju and P. Kivimaki, "Co-operation and Comparison of Diffserv and Intserv: Performance Measurements," *25th Annual IEEE Conf. Local Comp. Nets.*, 2000, pp. 177–86.
[12] Z. Di and H. T. Mouftah, "Performance Evaluation of Per-Hop Forwarding Behaviors in the DiffServ Internet," *5th IEEE Symp. Comp. and Commun.*, 2000, pp. 334–39.

Biographies

YING-DAR LIN [M] (ydlin@cis.nctu.edu.tw) received his Bachelor's degree in computer science and information engineering from National Taiwan University in 1988, and M.S. and Ph.D. degrees in computer science from the University of California at Los Angeles in 1990 and 1993, respectively. He is currently a professor of the Department of Computer and Information Science at National Chiao Tung University. His research interests include design, analysis, and implementation of network protocols and algorithms, wire speed switching and routing, quality of services, network security, and content networking.

YI-NENG LIN [M] (ynlin@cis.nctu.edu.tw) received his Bachelor's and Master's degrees in computer and information science from National Chiao-Tung University in 1999 and 2001, respectively, and is pursuing a Ph.D. degree in the same department. He was a member of a research and implementation project in Computer and Communication Laboratories, ITRI, in 2000 and 2001. His research interests include design and analysis of multithreaded multiprocessor architectures, and hacking of Linux and NetBSD kernels.

SHUN-CHIN YANG [M] (sc_yang@itri.org.tw) received his Bachelor's and Master's degrees in electrical engineering from National Chung-Cheng University in 1998 and 1999. His research interests include design and analysis of router and switch architectures, programming of Linux driver and kernel, and embedded system porting.

YU-SHENG LIN (yslin@itri.org.tw) received his Ph.D. in electrical engineering from National Chiao Tung University in 1998. Since 1998 he has worked as an R&D engineer with Communications and Computer Research Laboratories of the Industrial Technology Research Institute (CCL/ITRI), Taiwan. His research areas include high-speed packet switching architectures and implementation, and network security.