DiffServ over Network Processors: Implementation and Evaluation[†]

Ying-Dar Lin, Yi-Neng Lin
Department of Computer and Information Science, National Chiao Tung University, Hsinchu, Taiwan
{ydlin, ynlin}@cis.nctu.edu.tw

Shun-Chin Yang, and Yu-Sheng Lin
Computer and Communications Laboratories, Industrial Technology Research Institute, Hsinchu, Taiwan
{890047, 870932}@itri.org.tw

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 though the system can scale to wire-speed of 1.8Gbps in simple IP forwarding, the throughput declines to 180Mbps~290Mbps 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 result reported here shall remain the same for other NPs since they have similar architectures and components.

1. Introduction

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 firewall, DiffServ and Web switching. The traditional general-purpose processor architecture is no longer sufficiently scalable for wire-speed processing, and some 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 the acceleration cards for encryption/decryption, VPN gateways, Layer 3 switches, DiffServ routers and Web switches. While accelerating the 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.

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 much SRAM table accesses and computing power. This work therefore aims to implement a more sophisticated service, Differentiated Services (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, how well can this DiffServ implementation *scale* in terms of throughput for different numbers of classification rules? The reason we evaluate the scalability in terms of the number of classification rules is that since the number of classes in DiffServ is limited, it equals the number of flows, which is, in a sense, the number of classification rules. Second, where are the potential *bottlenecks* and their causes? The exact bottleneck is anticipated to depend on the specific *service* and its *algorithmic* implementation.

The rest of this paper is organized as follows. Section II briefly reviews the architecture of IXP1200. Section III then presents the design and implementation of DiffServ

[†] This work was supported in part by the MOE Program of Excellence Research 89-E-FA04-1-4 and a grant from Intel. The IXP 1200 platform was donated by Intel for development and evaluation.

over IXP1200. Next, Section IV illustrates the results of external and internal benchmarks through experiment and simulation, respectively. Conclusions are finally made in Section V.

2. Architecture of IXP1200

Closely examining the hardware architecture of IXP1200, shown in Fig. 1, helps to elucidate our DiffServ implementation. The 32-bit, 200MHz 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 200MHz 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 for storing lookup *tables* and *pointers* in scheduling queues for packet forwarding, while the SDRAM is used for storing mass data of *packets*. The 64-bit IX bus Interface Unit is responsible for servicing MAC interface ports on the IX Bus, and moving data to and from the Receive and Transmit FIFOs. It provides a 4.2Gbps (64bit at 66MHz) interface to MAC devices, meaning that it can afford 2.1Gbps of the input ports and 2.1Gbps of the output ports. In addition, two IXP1200 network processors can be directly supported on the IX Bus without additional support logic.

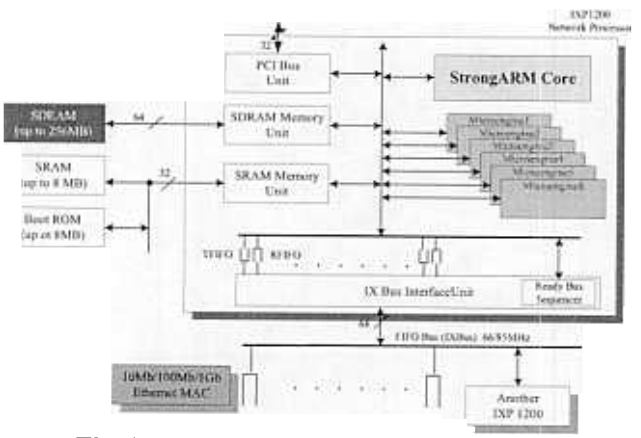

Fig. 1 Hardware architecture of IXP1200

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 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 TFIFO to another MAC. Multiple receiver, transmitter, and scheduler threads may be distributed to six microengines, although some restrictions apply.

3. Design and Implementation of DiffServ on IXP1200

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

3.1. DiffServ Briefing

Differentiated Services (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 (MF), leaky bucket policed, and marked to receive a particular per-hop forwarding behavior (PHB), which defines how packets with a particular behavior are treated at this node. Each predefined PHB is mapped to one DSCP (DiffServ Code Point) value used in class-based scheduling, i.e., Expedited Forwarding (EF) or one of four Assured Forwarding's (AF's).

The service differentiation of packets is often manifest 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.

Fig. 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 packet, referred as MP which is a basic data unit in the system, from the MAC buffer into RFIFO.

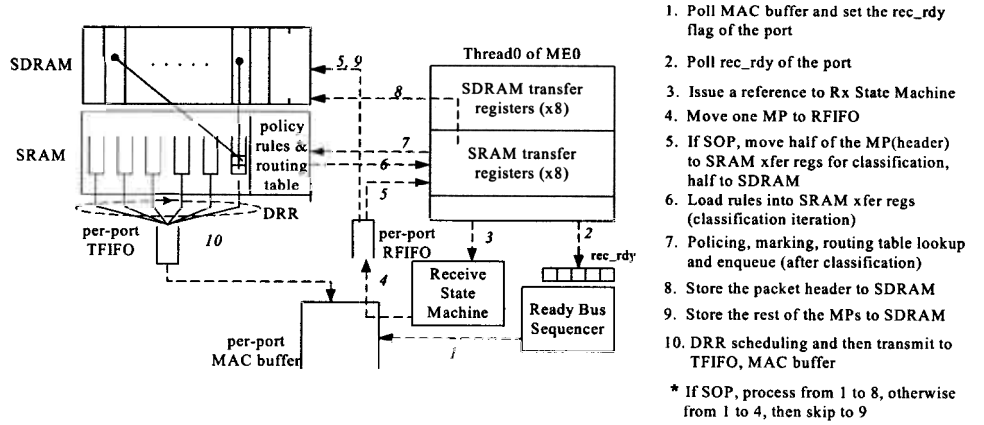

Fig. 2. Detailed DiffServ packet flow in IXP1200

3.3. Mapping DiffServ Components

Fig. 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.

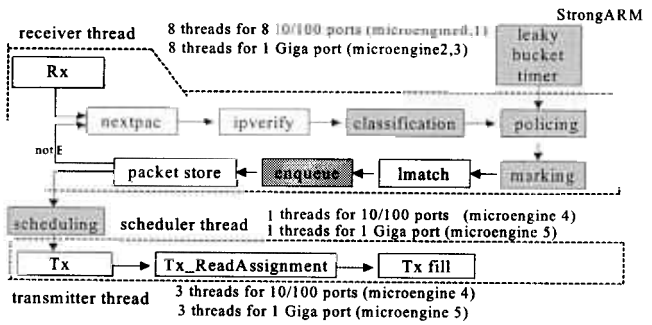

Fig. 3 Data-plane architecture of DiffServ edge router over IXP1200

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 (1 EF, 4 AF's and 1 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 having enough quanta, then the transmitter thread fetches the packet's descriptor in SRAM and then 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 twenty four threads are equally divided into two groups, eight 10/100M ports and one giga port. Each group has twelve threads: eight of which are used as receivers (assigned to two microengines), three of which are used 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 giga receiver threads serve one giga 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 1K 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 co-exist 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.

3.4. Algorithm Adoption and Implementation

3.4.1. Related Works. Classification and scheduling have been 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 (2Mbytes of SRAM) and coding overhead. For example, Recursive Flow Classification [4] which has an unstable memory size requirement ranging from 1Mbytes to 1000Mbytes; 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. Nontheless, 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 Queueing [6], which involves complex multiplications.

Multi-dimensional 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 easily implemented (requiring only *addition*) and effectively ensures weighted sharing among various flows. The following two subsections briefly describe the implementation of these two algorithms.

3.4.2. Classifier. The concept of Multi-dimensional Range Matching used to implement the classifier is described below. The rules in a dimension form intervals, in which multiple rules may overlap. Each interval is associated with a BV (Bit Vector, which is 512-bit in this implementation and is stored in SRAM), which keeps track of what rules overlap in this interval. The space complexity is O(n^2), where n is the number of classification rules, since there are at most 2n-1 intervals, and each of which keeps track of the n rules.

Fig. 4 presents an example of the matching process in the source IP dimension. When a packet arrives, the classifier searches the interval table of each dimension for a match with the corresponding field in the packet. When an interval is found for a dimension, the classifier retrieves the corresponding BV in the BV table. If all six fields of a packet match an interval in six dimensions, the classifier ANDs the BVs of the intervals, and the index of the first non-zero bit in the result vector becomes the index of the matched classification rule.

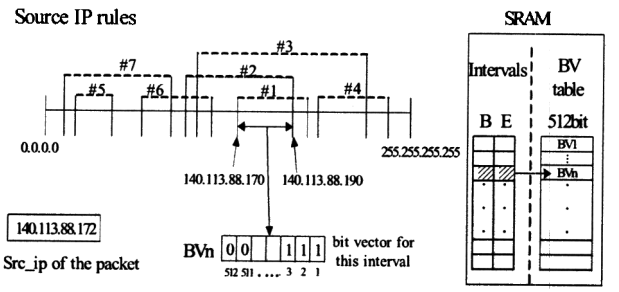

Fig. 4. Example and relative tables for lookup in Source IP dimension

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 in the processing of the last packet and is available for the next one. The total tokens available to the incoming packet can thus be determined and a decision regarding its admission can be made.

3.4.3. Scheduler. The quantum size of each class can be set arbitrarily. Here, we set the ratio of quantum between two adjacent classes in this system to two for simplicity. A packet is represented as a queue descriptor when queued in SRAM. Each queue descriptor contains the count Mac Packet (MP) and points to a link list of buffer descriptors, which point to the MPs of the packet stored in the 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 EOP (End of Packet).

4. External and Internal Benchmarks

The performance of DiffServ has been evaluated in a number of studies [9, 10, 11, 12]. However, most of these involve only simulations. Accordingly, this section considers two kinds of experiments, external and internal benchmarks. Firstly, the aggregated throughput that can be scaled by the system, while conforming the PHBs, is examiend to determine scalability. A Linear Search classifier is also included for comparison with the Range Matching classifier.

The internal benchmark involves simulations of two DiffServ implementations on IXP1200 whose classifiers are implemented with Linear Search and Range Matching. The aim is to identify what cannot be seen in the external benchmarks, and the performance bottlenecks. The number of classification rules is 64 with a worst case configuration with respect to the number of classification rule matching for both simulations.

4.1. Scalability Test

4.1.1. Traffic Load. The methodology for testing fairness between input flows is described below. The maximum load, which is 58%, is first measured for a flow that yields no packet loss. The fairness of the system for more flows is then obtained for input loads below or above 58%. Each flow is set to have the same bandwidth, and the aggregated bandwidth is 50% of the link. The input load is evenly distributed over 500 flows in the following test.

Fig. 5 presents the throughputs of 500 flows under two load conditions, with each flow exactly matches one of the 500 classification rules. The flows strictly follow their bandwidth settings when the input load is 50% and become unstable when overloaded. However, most of the flows are limited to their bandwidth settings.

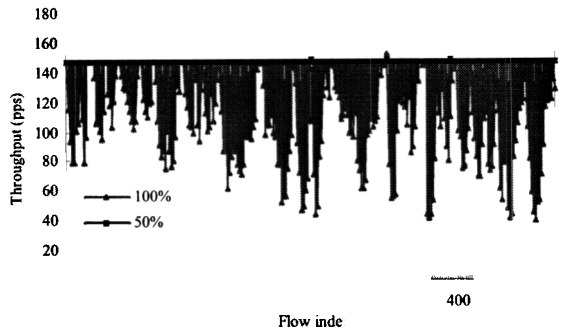

Fig. 5. Flow fairness test (Len=64bytes, 500 flows, BW=74400/500=148pps, average case)

4.1.2. Number of Microengines. Fig. 6 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 180Mbps to 290Mbps, 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.

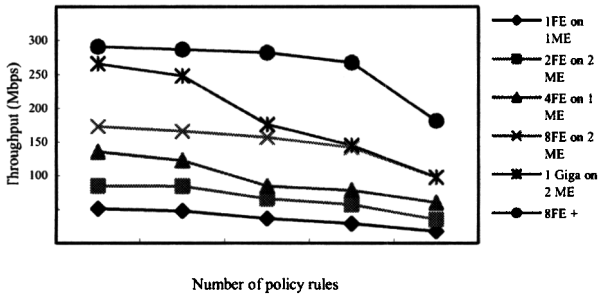

Fig. 6. Aggregated throughput (Len=64bytes, worst case)

Fig. 6 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 but 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 100Mbps input ports, whose threads are all in the same microengine, the bottleneck becomes the microengine because of a lack of computing power. 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 100Mbps 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.

4.2. Simulation- -with Linear Search classifier

The simulation result of the Linear Search classifier yields two observations. First is the low bandwidth utilization of SDRAM, sinice packet forwarding, which is the main consumer of SDRAM, is not essential in DiffServ. Second, both receiver microengines and SRAM are not fully utilized, but are 80% and 55% utilized, respectively, when the actual throughput of the system is low.

The following explanation applies. Although the utilization of SRAM is only 55%, it is a bottleneck because the SRAM access in the Linear Search classifier is *bursty*, meaning that the bandwidth of SRAM is not used until bursty access from microengines. Furthermore, sometimes all the threads in a microengine wait for SRAM access and thus cause the microengine to be idle.

4.3. Simulation—with Range Matching classifier

The utilizations of SDRAM and SRAM are again low at 13% and 35.3%, respectively, which result can be explained as for the Linear Search classifier. However, the receiver microengines are almost fully utilized in this simulation. Computing power can be identified as a performance bottleneck that leads to slow classification in the Range Matching DiffServ since both SRAM access and computing power are critical to the classification process.

5. Conclusions

This study explains the needs of network processors in today's complex applications, and introduces the architecture and packet flow in IXP1200. The mapping of DiffServ onto IXP1200 is detailed. DiffServ includes two very important modules, classifier and scheduler, which are implemented with Multi-dimensional Range Matching and Deficit Round Robin. Finally, external and internal benchmarks were applied to determine the bottlenecks in the implementation. Most of the result reported here shall

remain the same for other NPs since they have similar architectures and components.

The external benchmarks, which are in terms of number of rules or flows, traffic load and number of microengines, have established that the implementation can well support PHBs in DiffServ at an aggregated throughput of 568kpps. 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 not necessarily to occur at 100% utilization, but can even occur at 55% when the access is bursty.

Four methods are presented to solve the bottleneck of SRAM access that results in a 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, for example, QDR (Quad Data Rate) SRAM which has a peak bandwidth of up to 1.6GBps per channel (which is 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, CAM) 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.

6. Reference

[1] IXP1200 Data Sheet, Intel document number 278298-004, May 2000.

[2] T. Spalink, S. Karlin, L. Peterson, and Y. Gottlieb, "Building a Robust Software-Based Router Using Network Processors", *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP)*.

[3] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services", RFC 2475, Dec 1998.

[4] P. Gupta, and N. McKeown, "Packet Classification on Multiple Fields", *ACM SIGCOMM'99*.

[5] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel, "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 Multi-dimensional Range Matching", *ACM SIGCOMM'98*.

[8] M. Shreedhar, and G. Varghese, "Efficient Fair Queuing Using Deficit Round-Robin", *IEEE/ACM Transactions on Networking*, June 1996, vol. 4, no. 3, pp. 375-385.

[9] L.V. Nguyen, T. Eyers, and J.F. Chicharo, "Differentiated Service Performance Analysis", *Fifth IEEE Symposium on Computers and Communications*, 2000, pp. 328 –333.

[10] J.K. Muppala, T. Bancherdvanich, and A. Tyagi, "VoIP Performance on Differentiated Services Enabled Network", *IEEE International Conference on Network*, 2000, pp. 419 – 423.

[11] J. Harju, and P. Kivimaki, "Co-operation and Comparison of Diffserv and Intserv: performance measurements", *25th Annual IEEE Conference on Local Computer Networks*, 2000, pp. 177 –186.

[12] Z. Di, and H.T. Mouftah, "Performance Evaluation of Per-Hop Forwarding Behaviors in the DiffServ Internet", *Fifth IEEE Symposium on Computers and Communications*, 2000, pp. 334-339.

0-7695-1650-5/02 $17.00 © 2002 IEEE