Highspeed Packet Processing for DiffServ-over-MPLS TE on Network Processor

Djakhongir Siradjev[a], Youngsu Chae[b], Young-Tak Kim[c]
Yeungnam University, Korea

Abstract

The paper proposes an implementation architecture of DiffServ-over-MPLS traffic engineering (TE) on Intel IXP2400 network processor using Intel IXA SDK 4.0 Framework. Program architecture and functions are described. Also fast and scalable range-match classification scheme is proposed for DiffServ-over-MPLS TE that has been integrated with functional blocks from Intel Microblocks library. Performance test shows that application can process packets at approximate data rate of 3.5 Gbps. The proposed implementation architecture of DiffServ-over-MPLS TE on Network processor can provide guaranteed QoS on high-speed next generation Internet, while being flexible and easily modifiable.

Keywords: QoS, traffic engineering, network processor, fast packet processing, classification, DiffServ-over-MPLS

1. Introduction

The major goal of Internet traffic engineering is to provide efficient and reliable network operations and guaranteed Quality of Service (QoS) for end users, while keeping network resource utilization at high level. Support of Differentiated Services [1] in MPLS [2] that was standardized by IETF can provide QoS guaranteed service, while keeping network resource utilization at high level.

Unfortunately using DiffServ-over-MPLS [3] traffic engineering severely increases the processing complexity performed by the network nodes, especially edge routers. Also, taking into consideration of continuous exponential grow of network speeds, it is definite that improving the algorithms of packet processing together with parallelization of processing is one of the most important issues that should be studied.

For high-speed packet processing Network Processors are used [4]. There is a number of different implementations of DiffServ and MPLS on network processor, and moreover Intel supplies sample example applications [4], but there are no exact proposals of DiffServ-over-MPLS capable router on network processor. Other approaches, like implementations of DiffServ-over-MPLS routers do not have enough flexibility to be changed easily, when it is necessary, and implementations on General Purpose Processors do not have enough processing speed.

This paper concentrates on the architecture, implementation issues and techniques of DiffServ-over-MPLS router, and especially, range-match classifier microblock, on Intel IXP 2400 processor. We explain the design of processing pipeline and will go in depths of classifier architecture.

The rest of this paper is organized as follows. Section 2 describes the related works on Intel IXP 2400, Intel IXA SDK Framework, and DiffServ-over-MPLS traffic engineering. In Section 3, we describe the proposed scheme, implementation technique and issues for DiffServ-over-MPLS router, and its functional blocks. We will cover range-match classification implementation plan in details, as one of the main focuses of this paper. Section 4 analyzes the overall consideration factors of the proposed implementation scheme, and finally section 5 concludes the paper.

2. Related Works

2.1 Network Processors

Currently, most network interfaces of high-speed IP/MPLS routers are at 1 Gbps/2.4 Gbps (OC-48) ~ 10 Gbps (OC-192) using Gigabit Ethernet or SONET transmission links. In future, by using WDM and DWDM, 16 ~ 64 lambda channels (which can support 10 Gbps per lambda channel) are expected to be provided in an optical fiber, where IP/MPLS routers are going to support 8 ~ 64 optical links/ports. In order to support very high-speed packet processing and switching in total capacity of more than 1 Tbps, network processors with multiple microengines have been developed.

The IXP2400 network processor contains 2 kinds of processors: Intel XScale core and 8 microengines. Intel XScale core is a general-purpose, 32-bit RISC processor compatible to ARM Version 5 Architecture [4]. The Intel XScale core initializes and manages the chip, and can be used for higher layer network processing tasks. Microengines (MEs) are 32-bit programmable engines specialized for network processing; these MEs handle the main data plane processing per packet. There are eight

[a]Dept. of. Information and Communication Engineering
Graduate school, Yeungnam University
214-1, Dae-Dong, Kyungsan-Si, Kyungbook, 712-749, Korea
m0446086@chunma.yu.ac.kr
[b]Dept. of. Information and Communication Engineering
Graduate school, Yeungnam University
214-1, Dae-Dong, Kyungsan-Si, Kyungbook, 712-749, Korea
yschae@yu.ac.kr
[c]Dept. of. Information and Communication Engineering
Graduate school, Yeungnam University
214-1, Dae-Dong, Kyungsan-Si, Kyungbook, 712-749, Korea
ytkim@yu.ac.kr
This research was supported by the ITRC (Information Technology Research Center) program of the Ministry of Information and Communication , Korea

hardware threads per ME with no overhead for context switching. Also each microengine has local memory that can store up to 640 32-bit longwords, 256 general-purpose registers, 512 transfer registers, 128 next-neighbor registers that has hardware support for rings, and additional features, like content-addressable memory and multiplier.

The IXP2400 network processor provides a variety of different tools for implementing packet processing oriented applications, but has certain hardware limits, that written application can never exceed. That is why efficient application design is the key point of network processing applications.

2.2 Intel Internet Exchange Architecture Software Development Kit

The IXA Portability Framework [5] is a network application framework and infrastructure for writing modular and portable code, which can save time by providing robust infrastructure software and APIs, re-configurable building blocks, and an ideal structure for third-party plug-in application modules. Usual networking application usually contains data, control and management planes. The data plane processes and forwards packets at high speed. It is typically the most performance-sensitive since all packets processed by the device must pass through the data plane. Intel IXA SDK architecture splits the data plane into 2 parts: (i) the fast path which handle most of the packets, and (ii) the slow path which handles a few packets that cannot be handled on the fast path because of the complexity of the processing involved. These packets are called exception packets. Examples include forwarding IP packets with options in the header, handling fragmented packets, etc.

Every application built with IXA SDK Framework usually contains at least four logical parts: Microblocks, Core Components, System Application, and Dispatch Loops. Data plane processing on the Microengines can be pipelined by dividing the processing into tasks called Microblocks. The intent is to allow microblocks to be written so that each microblock is independent of the others. This improves reusability and allows developers to combine microblocks in different ways to create a meaningful application. Several Microblocks can be combined into a Microblock group using dispatch loop. A Microblock group runs on each thread of one or more Microengines.

The microblocks are usually one of two types: (i) Packet processing microblocks which perform high-level protocol-specific functions on a packet, for example IPv4 forwarding, MPLS, and Classification, and (ii) Driver microblocks which are hardware- or media-specific code blocks that may be implemented in hardware in future revisions of the IXP processor, i.e. Receive/Transmit blocks that interact closely with the MSF interface on the IXP2xxx.

2.3 DiffServ-over-MPLS

Differentiated Services [1] and Multiprotocol label switching (MPLS) [2] have been proposed and developed individually, and each of them has its specific goals and application fields. But it is possible to combine these two schemes in order to integrate their merits. DiffServ can provide microscopic flow control, by using differentiated per-hop behavior for each class type, while the MPLS traffic engineering provides macroscopic traffic engineering, like bandwidth reservation, and fault restoration. DiffServ-over-MPLS TE is standardized in IETF and allows end-to-end QoS provisioning.

Flows can be mapped onto LSPs in 2 different models:

- E-LSP (Exp-inferred-LSP) – an MPLS LSP can transport multiple class-types, and the EXP field of the MPLS Shim header defines the class type of packet together with drop precedence as shown in Fig. 1. E-LSP can carry only 8 different class-types of per-hop behaviors. Thus, E-LSP is usually used when class types and number of precedences is not so many, e.g. 4 class types and 2 drop precedences.

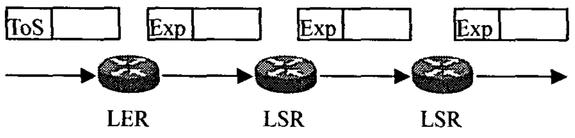


Fig. 1. Exp-inferred-LSP

- L-LSP (Label-only-inferred LSPs) –each LSP only transports a single class-type, so the packet label value defines class type, and the packet drop precedence is conveyed in the EXP field of the MPLS shim header as shown in Fig. 2.

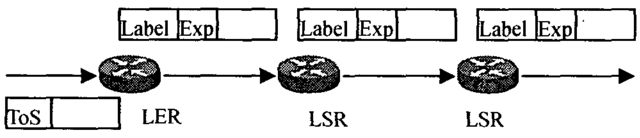


Fig. 2. Label-only-inferred-LSP

2.4 Range-match Classification

Classification is one of the main parts of DiffServ-over-MPLS processing and is necessary to split an incoming packet flow into several microflows to provide differentiated per-hop behaviour, and assign an MPLS label according to Forwarding Equivalence Class (FEC). The exact match classifier provided by Intel [6] is a fast solution for classification. Its design is very simple, and is based on using hashing. When a new rule is created a hash value is calculated from necessary fields of packet header. Then hash is used as an index to classification table entry that is filled with necessary information. At each packet arrival, hash value from same header fields is calculated and the value is used to access the classification information. The exact match classifier can be used for some packet processing applications, but DiffServ MIB requires range-match classification. That's why for full-featured DiffServ-over-MPLS application range-match classifier has to be implemented.

Range-match classification is found to be one of the most complex packet-processing procedures in time-memory complexity. Most practical solutions use linear time to search through all rules sequentially, or use a linear amount of parallelism, e.g., CAM (Content Addressable memory). While Ternary-CAMs are very common, such CAMs have smaller density than standard memories, dissipate more power, and require multiple entries to handle rules that specify ranges. Thus, CAM solutions are still expensive for very large rule sets of, say, 100 000 rules, and are not practical for PC-based routers [7]. Other solutions also do not show good results and usually have very high time or memory complexity. This generates a big problem of implementation of range-match classification that can work on wire speed of more than 1 Gbps, and does not require more resources than network processor has. Solution provided in [7] is one of the algorithms that can fit the requirements of network processor. The idea is based on Bit Vector scheme [8], which works according to following procedure:

- k one-dimensional tries, associated with each dimensions in the original database, are built, where k is the number of fields designated for classifying.
- an N bit vector is associated with each node of the trie corresponding to a valid prefix, where N is the maximum number of rules supported by classifier.
- after searching in each dimension, bitwise AND operation is performed on them and the final bit mask of matching rules is found.

But with large number of rules the size of bit vector will increase, cause a scalability problem. Baboescu et. al. [7] proposed new scheme called Aggregated Bit Vector (ABV) algorithm which uses the hierarchy of bit vectors to reduce a lot of unnecessary memory accesses. But this algorithm requires longer preprocessing time, because, for better performance rules should be sorted within 1 or 2 fields. The sorting is done in following way:

- All rules are sorted according to prefix lengths in one of the fields.
- Within groups with same prefix lengths, rules are sorted by prefixes.
- The same operations are performed on other fields, but within groups with same prefixes in previous field.

The results provided in [7] shows that the algorithm has very good merits comparing to others, but it will have lower performance in case when interdependent prefixes are distributed over the table, and the proposed sorting in [7] can not provide the best case placement of rules in the table. In this paper we made several modifications to improve the classification algorithm performance, at the expense of rule database update performance.

3. Diffserv-over-MPLS TE on Network Processor

3.1 Program architecture

The implementation of DiffServ-over-MPLS TE is performed on IXDP 2400 development platform, which contains 2 IXP 2400 Network Processors connected using switch fabric loopback. The overall processing is split into 2 parts: ingress and egress. Ingress side processing is usually responsible for determining the next hop information, flow ID, LSP ID and class ID information, while egress side processing usually performs queuing and scheduling, traffic shaping, L2 header encapsulation and sending the packet to the network.

Fig. 3. shows the mapping of tasks in ingress side packet processing to microengines. Driver microblocks are usually mapped to a single microengine, while processing microblock group is mapped to all other free microengines. In order to improve the application performance, microengines for processing microblocks, were split into 2 groups in separate clusters. Packet order is maintained using ordered thread execution [9,10]. Different microblock groups are connected to each other using scratch rings and next-neighbor rings [9]. Full-scale queue manager and scheduler on ingress side are necessary only in multi-blade systems. In single-blade systems queue manager will maintain only one queue, and scheduler will need only to handle flow control messages from switching fabric and

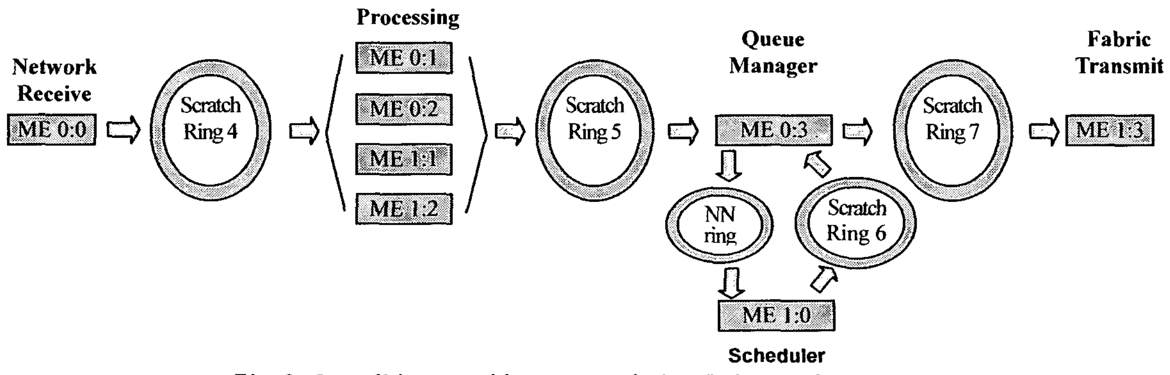


Fig. 3. Overall ingress side program design (Microengines)

send dequeue requests to queue manager. Fabric transmitter microblock transmits cells to CSIX interface. Each packet is split into a number of C-frames and transmitted to switching fabric. First C-frame of the packet carries additional information, such as packet color, LSP ID, class ID, and next-hop information.

The ingress processing pipeline is displayed on Fig. 4. After receiving a packet from scratch ring, the Ethernet decapsulation and classification microblock checks whether packet is IP or MPLS packet. If the packet is IP packet than it goes through header verification, and then passed to the classifier microblock. If the classifier (currently it is an exact match classifier) finds a match in its database, it assigns flow ID, next-hop ID and class ID, and sends packet to meter microblock that checks the conformance of the packet. Then, the packet's DiffServ codepoint (DSCP) is set, and it is sent to FTN (FEC to NHLFE) microblock that performs MPLS label stack operations. If classifier does not find any match, the packet is handled by simple best effort IPv4 forwarding.

When the incoming packet is an MPLS packet, then it will go through MPLS ILM microblock only and LSP ID will be set after label stack operations will be performed. Proposed architecture of processing pipeline will allow the implemented router to support the functionality of both edge and core routers.

Egress side application design is quite simple, and is mostly related with per-hob behavior (PHB) of the router as shown in Fig. 5. First PHB related microblocks is WRED (Weighted Random Early Detection) microblock. The Layer 2 encapsulation microblock from Intel was changed in order to avoid L2 table lookup when MPLS is used. When MPLS is used, simple Ethernet header with broadcast source, destination addresses and the MPLS packet type is added. The most complex part of the egress microblock is queue manager/scheduler combination of microblocks. The Queue Manager should maintain separate logical queues for each class-type of each E-LSP, and one queue per each L-LSP. Scheduling must be performed within each E-LSP, within a group of high priority L-LSPs and low priority L-LSPs. Although such architecture will increase the complexity of egress side, it is an optimal way to provide MPLS traffic engineering combined with DiffServ per-hop behaviors.

3.2 Implementation of range-match classifier

In the current status of the application, only the exact match classifier from Intel Microblocks library is used. Unfortunately, as mentioned in previous section, it cannot provide scalable packet classification, since most of rule databases of edge routers are using ranges in most of the fields. But the approach used in this microblock, which uses hash values for fast exact match lookup will be used in further.

The aggregated bit vector algorithm described in section 2.4 is found to be a good candidate for DiffServ-over-MPLS application being developed. The original Aggregated Bit Vector (ABV) proposal does not cover the method of search in each of fields describing only the

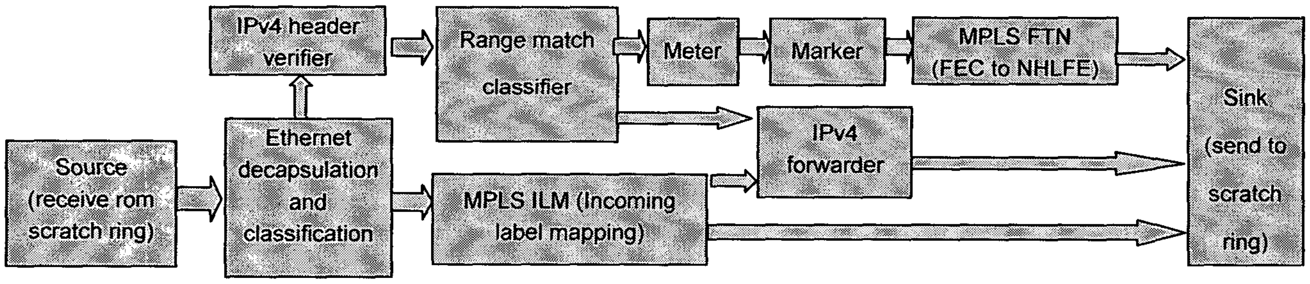

IPv4 header verifier
Range match classifier
Meter
Marker
MPLS FTN (FEC to NHLFE)
Sink (send to scratch ring)
Source (receive rom scratch ring)
Ethernet decapsulation and classification
IPv4 forwarder
MPLS ILM (Incoming label mapping)

Fig. 4. Design of ingress side processing pipeline

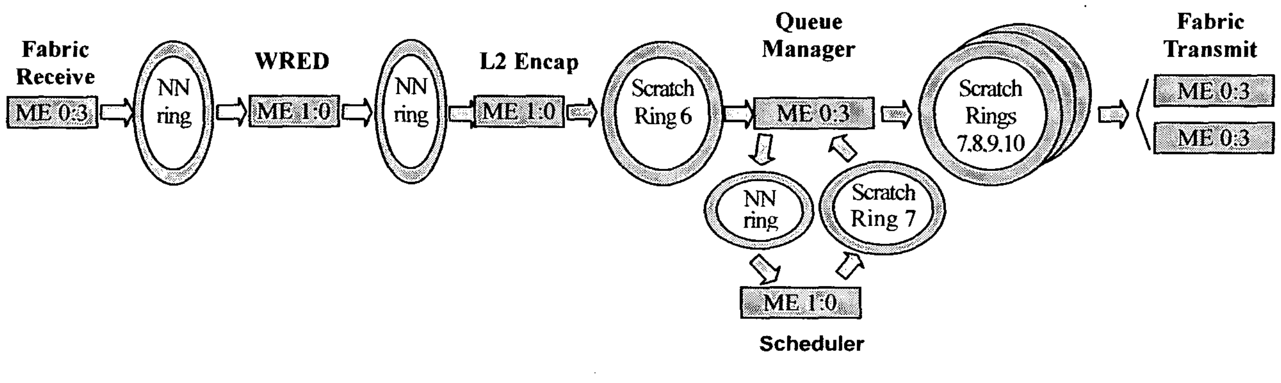


Fig. 5. Overall egress side program design (Microengines)

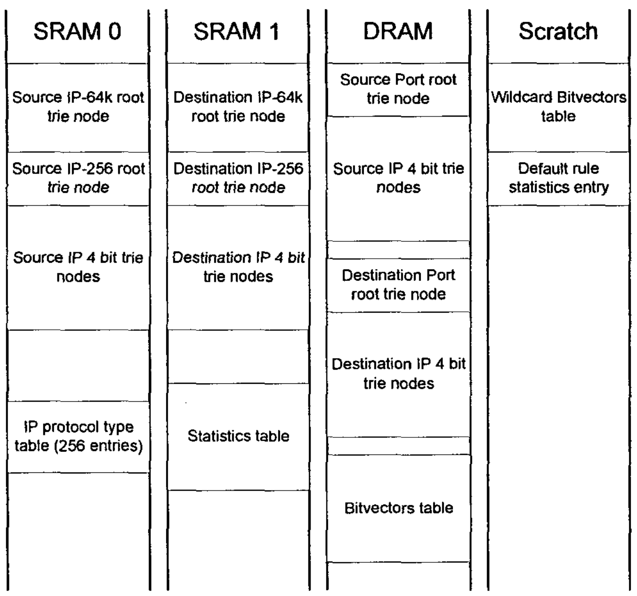

SRAM 0
Source IP-64k root trie node
Source IP-256 root trie node
Source IP 4 bit trie nodes
IP protocol type table (256 entries)
SRAM 1
Destination IP-64k root trie node
Destination IP-256 root trie node
Destination IP 4 bit trie nodes
Statistics table
DRAM
Source Port root trie node
Source IP 4 bit trie nodes
Destination Port root trie node
Destination IP 4 bit trie nodes
Bitvectors table
Scratch
Wildcard Bitvectors table
Default rule statistics entry

Fig. 6. Organization of data structures in memory

aggregation technique that exploits the sparseness of bit vectors. In our implementation we use multi-bit tries to decrease the search time in individual fields. Tries for lookup in IP source and destination addresses fields will be built in a same way as it was built for IPv4 forwarder microblock by Intel [6] using dual lookup algorithm. Port tries has root trie node representing 8-bits and child trie nodes representing 4 bits. For protocol lookup simple table is used, since exact-match lookup is sufficient.

To increase parallelization, search must be done in multiple fields in parallel. For that purpose, we propose a memory hierarchy where different tries are stored in different types of memory. Organization of the proposed data structures in memory is shown on Fig. 6.

To avoid fragmentation of memory on dynamic memory allocation and deallocation, simple memory management system is implemented. The idea is to allocate memory for maximum number of entries, which is system parameter depending on free memory size, once during program initialization. After that, numbers from 1 to maximum number of entries are inserted in one-directional linked list, called freelist. When memory for new entry is required, unused memory index is retrieved from freelist, and reverse operation is done on deallocation. This approach also helps to reduce memory used by entries, because instead of keeping 32-bit memory pointers they will just keep 16 bit indices.

The design of microblock exploits the serial operation of memory controller and special features microengines. So to perform parallel search in all fields, read operations of memory from all trie nodes is requested. After that, the microengine waits until any of reads is finished. Once any read is finished, microengine continue search requesting new read for certain field, and waiting on any signal. For reading partitions of bit vector all the reads are requested one after the other, and the signal indication completeness is expected only for the last read. This helps to avoid the limitation in number of signals.

Also there can be some enhancements in sorting algorithm which can increase the performance of classifying. The main goal of sorting is to group the rules that the same packet can match together, since this will improve the effect of aggregation. We suggest using lexicographic sorting instead of sorting by prefixes. Table 1 shows the example.

Table 1. Example of lexicographic vs. prefix-based sorting.

Grouping according to algorithm in [7]	Grouping according to our suggestion
*	*
*	*
*	*
150.*	150.*
150.*	150.*
170.*	150.200.*
150.200.*	162.200.*
162.200.*	162.200.*
162.200.*	170.*
170.10.23.*	170.10.23.*

As it can be seen from the table, the aggregation will have much higher performance improvement in the second case, than in the first one, which was proposed in [7]. But this approach has a disadvantage in preprocessing time, since lexicographic sorting requires longer time. Usage of this approach will depend on certain circumstances, which the application will be used under.

3.3 Implementation details on other DiffServ related microblocks

For Diffserv related tasks, microblocks from Intel Microblock Library with minor changes, are used. The three color metering (TCM) is standardized by IETF [12, 13]. In the implementation TCM microblock from Intel Microblocks Library has been used. It supports both Single-Rate TCM and Two-Rate TCM. Each flow is metered separately, and flows are distinguished by using flow ID, set by classifier.

Periodic token updates, required by standard, bring in performance concerns in IXP2400 and IXP2800 Network Processors. Basically, such solution would require either a dedicated microengine thread that continuously updates the metering table entries, or a background task running on Intel XScale core. The first solution is not applicable since the SRTCM block is a part of a functional pipe stage. All threads constituting a functional pipeline have to perform the same operations. On the contrary, a dedicated task on Intel XScale core would have to inspect meter entries quite frequently due to possibly high flow rates. As a result, a significant amount of SRAM memory accesses would occur, which is critical to the fast path performance. So the token counts are updated by using interval between current and previous timestamps multiplied by token count update rate (CIR or PIR). TCM microblock takes 109 cycles to process a packet in the worst case.

The DSCP marker microblock is quite simple. It has to set up the DiffServ codepoint in IP packet header. The class ID value is determined by classifier, and color ID is determined by meter. The update of checksum is performed by the method given in [14].

4. Performance Evaluations and Analysis

4.1 Performance evaluation

Fig. 7 depicts the testbed topology for performance measurement and evaluation. For the traffic generating purposes SmartBits SMB-6000 traffic generator with 2 ports was used. 2 IXDP 2400 Development platforms are used to run the application. In order to test the capability of processing of 4 Gbps throughput, IXDP 2400 platforms were connected to each other several times (Fig. 8.), so each port sends and receives 1 Gbps streams at a time. Exact-match classifier was used in the measurement, because range-match classifier is under implementation stage. The estimated performance of the proposed range-match classifier is provided in section 4.3

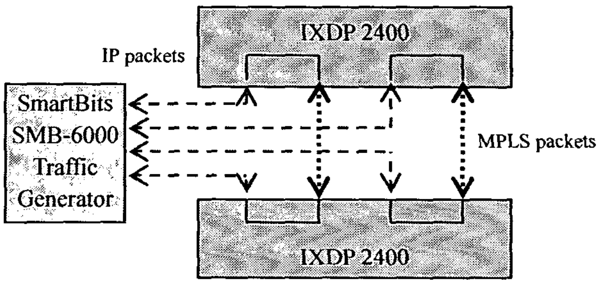


Fig. 7. Testbed topology

4.2 Analysis of the Results

The designed DiffServ-over-MPLS packet processing modules show on the average 3.5 Gbps throughput (not considering encapsulated MPLS shim headers), as shown in

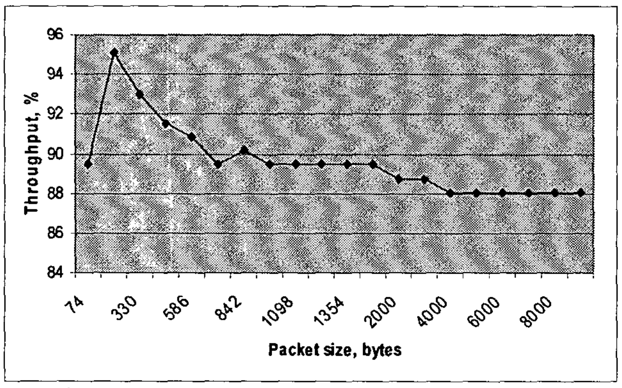


Fig. 8. Throughut vs. packet size (Load=4 Gbps)

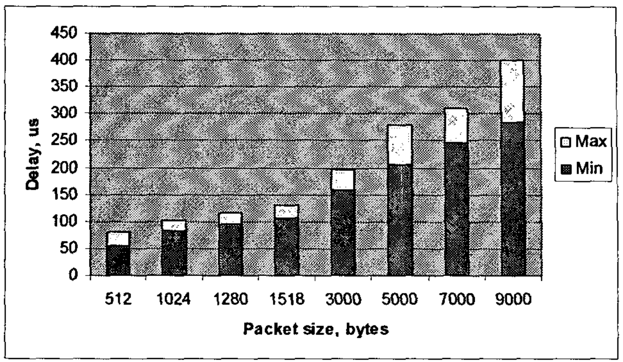


Fig. 9. Average latency vs. packet size

Fig. 8. The low throughput in the range of the very small packet size (78 bytes), is described with the big number of packets since each of packets will have additional shim header added, and not enough processing power to process so large number of packets. Decreased throughput with the increased packet size is caused by lack of receiving buffers in MSF interface, since the bigger sized packets will require more space in the buffers and longer time necessary to store the whole packet before the processing will start.

The average packet delay was also measured using the same topology. The results are shown in Fig. 9. Since the application is using store-and-forward technology, the delay increases when the packet size increases. Although the implemented application supports jumbo frames, the performance of jumbo frames processing is a little worse than the case of non-jumbo frames. The implemented MPLS processing supports up to 4 label pushes, and recursive label processing on incoming MPLS packets. Microblocks taking the longest processing time is IPv4 forwarding, because it requires the longest prefix match lookup to be done. Once range-match classifier is implemented it will be the microblock taking the longest processing time, because it will perform lookup several times.

4.3 Analysis of range-match classifier

Since the range-match classifier is under implementation stage, currently only expected performance results are available. The only performance critical part of classifier is microblock. As it can be seen from pseudocode, the microblock requires at most 5 consecutive memory accesses for searching all fields, since it is the longest search time among all fields and all lookups are done in parallel. The last access here is reading of ABV. After ABV operations are completed one more burst of accesses is necessary for corresponding partitions of bit vector. So, the longest set of consecutive memory accesses is: 4 SRAM + 2 DRAM accesses, which will result in (4*90+2*120) =600 cycles. Adding here the rule entry read we get 1 more SRAM access, so the minimum number of cycles required for classification will be 690 cycles, while totally around 3000 cycles are allowed for overall processing.

Total number of cycles required for classification increases, depending on the microengine instructions number in classification, load of memory channels and organization of pipelining in memory controller. Usage of statistics also increases the processing time. Since 64 bits are used as aggregation size and word size, maximum number of rules supported is 64x64=4096.

4.3 Scalability

Theoretically the system can be designed with maximum number of 16 blades, with 4 ports on each blade. The number of flows is limited by 4096, because 64 bit word size and aggregation size is chosen according to the largest memory width available in Intel IXP 2400 network processor. The number of total possible LSPs and next hops of IP routing table can reach up to 2^{16} (may be less, depending on queue manager restriction configuration). Labels can be assigned in per-interface and per-system manner. The practical limitations can be lower depending on the memory size of used platform.

5. Conclusion

In this paper, we analyzed the implementation details of DiffServ-over-MPLS architecture on Network Processor and particularly the classification problem. We proposed an improved classification algorithm, based on Aggregated Bit Vector scheme. Application in its current status shows good results in performance test. Although the DiffServ-over-MPLS architecture is well-known, most of its implementations were designed for hardware, while the implementation on network processor can make it more flexible. The proposed DiffServ-over-MPLS TE implementation on network processor should help in the analysis of functional blocks and improving them in future. Future works include implementation of the proposed range-match classifier architecture, dynamic queue manager/scheduler, functional modules of control plane and OAM functionality.

References

[1] S. Blake et al., “An Architecture of Differentiated Services,” RFC 2475, IETF, December, 1998.

[2] E. Rosen et al., “Multiprotocol Label Switching Architecture,” RFC 3031, IETF, January, 2001.

[3] F. Le Faucheur, editor, “Multiprotocol Label Switching (MPLS) support of Differentiated Services,” RFC 3270, IETF, April, 2002.

[4] “*Intel® IXP2400 Network Processor Hardware Reference Manual, Intel Corporation,*” October, 2004.

[5] “*Intel® Internet Exchange Architecture Portability Framework Developer's Manual,*” Intel Corporation, November, 2004.

[6] “*Intel® Internet Exchange Architecture Software Building Blocks Developer's Manual,*” Intel Corporation, November, 2004.

[7] Florin Baboescu and George Varghese, “Scalable Packet Classification,” IEEE/ACM transactions on networking, vol. 13, No. 1, February 2005.

[8] T. V. Lakshman and D. Stidialis, “High speed policy-based packet forwarding using efficient multi-dimensional range matching,” in *Proc. ACM SIGCOMM*, Sep. 1998.

[9] Erik J. Johnson and Aaron R. Kunze, “*IXP2400/2800 Programming,*” Intel Press, 2003.

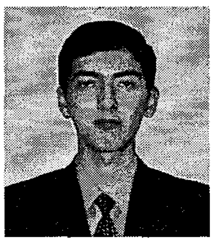
[10] Bill Carlson, “*Intel Internet Exchange Architecture and Applications,*” Intel Press, 2003.

[11] Y. Tung and H. Che, "Study of Flow Caching for Layer-4 Switching,” in the Proceedings of IEEE ICCCN’00, pp. 135-140, 2000.

[12] J. Heinanen, Telia Finland, R. Guerin, “A Single Rate Three Color Marker,” RFC 2697, IETF, September, 1999.

[13] J. Heinanen and R. Guerin, "A Two Rate Three Color Marker,” RFC 2698, September 1999.

[14] A. Rijsinghani, Editor, “Computation of the Internet Checksum via Incremental Update,” RFC 1624, IETF, May 1994.



Djakhongir Siradjev is a candidate of M.S. degree in the department of Information and Communication Engineering at Yeungnam University. His research focus is fast packet processing on network processors. He has investigated issues in implementing of DiffServ-over-MPLS related packet processing algorithms on network processors, provisioning of guaranteed Quality of Service (QoS). Mr. Siradjev has given presentations at regional and national conferences and workshops.

Youngsu Chae received the B.S. and the M.S. degrees in computer science from Pohang University of Science and Technology, Pohang, Korea, in 1994 and 1996, respectively. He is currently a faculty member of School of EECS, Yeungnam University, Kyoungsan, Korea. His research interests include mobility management and QoS support in mobile networks, large scale Internet service architecture, and 4G networks.

Young-Tak Kim is a professor in the school of electrical engineering and computer science (EECS) of Yeungnam University, Korea. He graduated Yeungnam Univ. in 1984, and received Master Degree and Ph.D. degree from KAIST (Korea Advanced Institute of Science and Technology) in 1986 and 1990, respectively. He joined

Korea Telecom (KT) in March 1990, where he had researched and developed an ATM Metropolitan Area Network (MAN) Switching System (ATM-MSS). He transferred to Yeungnam University in September 1994. He has performed many research projects in the area of "High-speed Telecommunications Networking" and "Network Operations and Management." Currently he is the director of government supported University IT Research Center (ITRC) which is developing "QoS-guaranteed Traffic Engineering and Multimedia Service Platform in the Broadband convergence Network (BcN)." His research interests include QoS-guaranteed inter-AS traffic engineering and broadband mobile Internet service provisioning on a broadband converged wired & wireless network environment. Prof. Young-Tak Kim is a member of IEEE Communication Society, KICS (Korea Institute of Communication Society), KISS (Korea Information Society), KIPS (Korea Information Processing Society), and Korea Multimedia Society. He has been working as an organization committee member of APNOMS (Asia Pacific Network Operations and Management Symposium), and IEEE NOMS-2004 (Network Operations and Management Symposium).