A New Hardware Algorithm for Fast IP Routing Targeting Programmable Routers

Some Considerations

Mahmoud Meribout[1] and Massato Motomura[2]
[1] ECE Department, College of Engineering, SQU University, Oman.
[2] ULSI Laboratories, NEC Corporation, Tokyo, Japan.

Abstract: While deployment of embedded and distributed network services imposes new demands for flexibility and programmability, IP address lookup has become significant performance bottleneck for the highest performance routers. Amid their vast array of academic and commercial solutions to the problem, few achieve a favorable balance of performance, efficiency, and cost. New commercial products utilize Content Addressable Memory (CAM) devices that achieve high lookup speeds and an exorbitantly high hardware cost with limited flexibility. In this paper a new IP forwarding hardware algorithm, based on gray-code encoding, along with its dedicated scalable lookup engine is proposed. The corresponding programmable router is able to achieve high performance with the use of a small portion of Dynamic Reconfigurable Logic (DRL) device and a commodity Random Access Memory (RAM) logic. Experimental evaluation using Ns2 simulator and a small routing table from Mae West [1] has been conducted. The results show that the programmable router can be scaled to achieve guaranteed worst-case performance of over 66 million lookups per second with a single SRAM operation at the fairly clock of 100 MHz.

Key words:IP forwarding, Internet Router, FPGA, DRL, Networking, distributed hardware.

1. INTRODUCTION

As physical link speeds grow and the number of ports in high-performance routers continues to increase, there is a growing need to efficient lookup algorithms and effective implementation of those algorithms. Next generation routers must be able to support thousands of optical links each operating at 10Gbps (OC-192) or more. Lookup techniques that can

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35703-4_21
D. Gaïti et al. (eds.), *Network Control and Engineering for QoS, Security and Mobility II*

scale efficiently to high speeds and large lookup table sizes are essential for meeting the growing performance demands while maintaining acceptable per-cost costs.

The most efficient lookup algorithm known, from a theoretical perspective, is the "binary search over prefix lengths" algorithm described in [8]. The number of steps required by this algorithm grows logarithmically in the length of the address, making it particularly attractive for IPv6, where address lengths increase to 128 bits. However, the algorithm is relatively complex to implement, making it more suitable for software implementation than hardware implementation.

From the hardware point of view, several companies have developed high-speed lookup techniques using CAM and ASICs [3, 4, 5, 6]. Some Products, targeting OC-768 (40Gbps) and OC-192 (10Gbps), claim throughputs of up to 100 million lookups per second and storage for 100 millions entries [7]. However, this comes at an extreme cost. 15 ASICS containing embedded CAMs must be cascaded in order to achieve the advertised throughput and support the more realistic storage capacity of one million table entries. Such exorbitant hardware resource requirements make these solutions prohibitively expensive and preclude System-On-Chip (SOC) port processors. Additionally, CAM-based lookup tables are expensive to update, since the insertion of a new routing prefix may require moving an unbounded number of existing entries. CAM approaches also offer little or no flexibility for adapting to new addressing and routing protocols.

In this paper, we present the results of a new fast IP forwarding hardware algorithm using our DRL-based platform [15]. Section 5 which report the performance of the evaluations using a snapshot of the Mae-West routing table resulted in 64 Millions lookups per second. This leads to consider our proposal as one potential candidate for next generation QoS-enabled programmable router architectures. An ongoing research seeks to exploit advanced CAD tool in order to double the clock frequency, therefore, double the lookup performance [15]. Another research effort leverages the insights and components produced.

2. PRIOR WORK AND MOTIVATION

IP forwarding of packets is one of the most predominant tasks required in a router. Its principle is to search the longest prefix (e.g. the larger number of bits, starting from the most significant bit (msb) to the least significant bit (lsb)) corresponding to a given destination IP address. The "binary search over prefix lengths" algorithm [8] has been widely used for this purpose. Its idea is to store prefixes in a binary trie. A search is conducted by using the

IP address bits to traverse the trie, starting with the most significant bit of the address (e.g. from the route of the trie). The router will then direct the packet to one of its outputs, which corresponds to the resulting prefix. This algorithm has the advantage to provide an optimal use of the forwarding table (e.g. a new next-hop entry can be stored at any address of the forwarding table memory). Its drawback however is the long search procedure and the high number of memory accesses equal to at least the number of bits in the destination IP address. This leads to a high complexity of the computation time to O (m) (where m is the number of bits in the destination IP address), making the algorithm more attractive for software implementation. Eartherton and Ditta's Tree-bitmap algorithm [10] is one of the latest algorithms which was designed to speedup the search procedure. This is achieved by simultaneously processing multiple bits of the destinations address. In [16], a successful hardware implementation of the algorithm onto a prototype router that uses the Xilinx's Viretex static Field Programmable Gate Array (FPGA) [11] as IP forwarding engine has been reported. The results indicated that the processing of each subtree requires a delay of 135 ns because of the long chain of arithmetic and logic operators along the critical path. The other drawback of the hardware algorithm is that whenever one single element of subtree, which is the child of a given root in the tree, is stored in the forwarding table, then all the 16 words of the forward table memory, which store the 16 next hop addresses of the same subtree, can't be used by other subtrees (e.g. subtrees with different root), even though the other prefixes of the actual subtree are not used. This leads to an under-optimal use of the forwarding table memory. In the next section, we will demonstrate how our new algorithm can overcome both these two disadvantages.

3. A NEW HARDWARE ALGORITHM FOR FAST IP FORWARDING

3.1 The New IP Forwarding Algorithm

The algorithm uses a forwarding table memory, R_{fwd}, to store either the next hop values or the intermediate addresses used to reach the next hop value. Its principle is to divide the destination IP address into n disjoint segments, (f_1, f_2,...f_n) of $\frac{m}{n} = \log_2(u) = q$ bits each, where m is the total number of bits in the destination IP address (which is equal to 32 and 128 for IPv4 and IPv6 respectively) and u the address bus width of the R_{fwd} memory. Additionally, we assume that the data bus width of the R_{fwd} memory is equal

to v. The algorithm then proceeds by hierarchically treating these segments starting from the most to least significant bits. The bits of each segment are treated in parallel which has the merit to reduce the number of memory accesses needed to perform lookup operations by the factor q (instead of m as in [8] and [10]). Each word of the forwarding table memory is divided into 4 fields (Figure 1).

First, let us take a look at writing a piece of text using a traditional typewriter.

- You insert a sheet of paper into the machine.
- You type your text exactly where you want it placed on the paper. When necessary, you move the writing head over to the desired location (usually by pressing the space bar repeatedly) before you press a character key.
- When you reach the end of a line, or whenever you wish (for any reason) to move to a new line, you use the carriage return lever as many times as you want to insert blank lines.
- The resulting print on paper is itself the text: the text does not exist independently of the print.

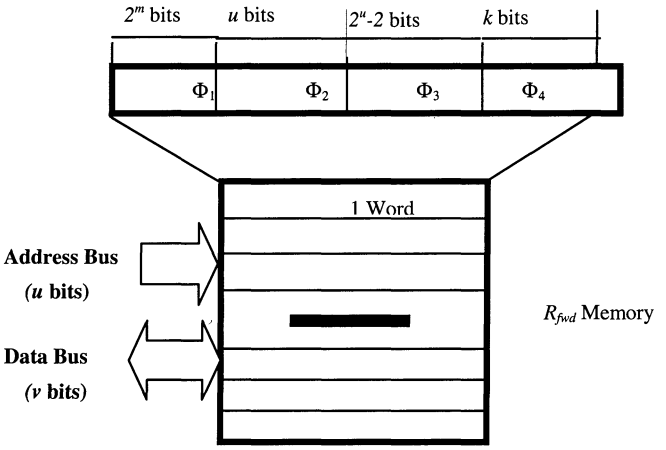

Figure 1 Data structure of the Forwarding table

The first field, Φ_1 (2^m bits) indicates if the actual segment is already stored in R_{fwd}. The second field, Φ_2 (u bits), indicates which bit of the actual address should be inversed. This depends on the value of the actual segment. The two other fields $\Phi_3((2^u\text{-}2)\text{-bits})$ and $\Phi_4(k\text{-bits})$ (where k is the number of hops available in the Internet router) are used to extract the longest prefix value and the next hop address respectively.

For each IP header, the algorithm starts by reading from the most to least significant segments. The information stored in the Φ_1 field indicates if the actual segment, f_i ,($i \in [1,n]$) being treated has or not a child node. In case a child node follows the input segment, then the actual R_{fwd} address, $Adr(f_{i-1})$,

which stores the information of the next segment, f_{i-1}, (e.g. Φ_1, Φ_2, Φ_3, and Φ_4 fields) is obtained by inversing the bit of the actual address of the forwarding table memory, $Adr(f_i)$, which is located at the $(r+1)^{th}$ most left "one" in the Φ_2 field corresponding to f_i segment. Here r is the total number of "ones" existing at the left of the bit located at position f_i. in its corresponding Φ_1 field:

$$Adr(f_{i-1}) = Adr(f_i) \oplus 1 << k \qquad (3.1)$$

$$\text{Where } k = \sum_{i=u}^{k} Bit_i(\Phi_2(f_i)) \qquad (3.2)$$

$Bit_i(\Phi_2(f_i))$ returns the value of bit "i" of the field Φ_2 that corresponds to the segment f_i.

In case the input segment does not have a child node, then the algorithm uses the information stored in the fields Φ_2 and Φ_3 to carry out the next-hop value, which corresponds to the longest prefix corresponding to the input address.

One of the advantages of our algorithm is that its data structure is small and could easily fit into a single SRAM chip memory. Furthermore, the algorithm can be efficiently mapped into hardware to provide Gigabit line speed since the intermediary addresses of the R_{fwd} memory are obtained by only inversing 1 bit of the previous address.

3.2 Walk Through Example

To better illustrate the algorithm, lets consider the case of m = 16 and u = 24 bits and in which a memory word of R_{fwd} contains the following fields:

Φ_1 = 1 1 1 0 0 1 0 1 1 0 0 0 1 0 0 1 and
Φ_2 = 0 0 0 0 0 0 0 0 1 1 0 0 1 0 1 0 0 1 1 1 1 0 0 0

This means that the segments whose values are equal to 16, 15, 14, 11, 9, 8, 4, and 1 already exist in the forwarding table. In addition, from the field Φ_2, we can deduce that the memory addresses corresponding to these segments can be obtained by inverting the bits 14^{th}, 13^{th}, 10^{th}, 8^{th}, 7^{th}, 6^{th}, 5^{th}, and 4^{th} respectively.

3.3 The IP Forwarding Table Updating Algorithm

During the updating of the forwarding table memory, R_{fwd}, the IP forwarding algorithm requires a special encoding scheme of the address bits

of R_{fwd},. It uses gray scale encoding to find the R_{fwd} address that shall store the information of the next segment (e.g. child node). The use of this encoding scheme allows having only one bit difference for the R_{fwd} addresses storing consecutive segments f_i and f_{i+1}. Hence, for a given IP address prefix, *IP_address_Prefix*, and next hop value, *nxt_hop*, the algorithm checks if the first segment, f_n, of *IP_address_Prefix* has already a child node in the R_{fwd} memory. In case it has not, the algorithm calculates the next unused address, $Adr(f_{n-1})$, (coded in Gray-code) which should store the information corresponding to the next segments, f_{n-1} of *IP_address_Prefix*. Hence, the first condition is that the transition from $Adr(f_n)$ to $Adr(f_{n-1})$ requires only one bit inversion. All the other bits remain unchangeable (Equation (3.3)). The second condition is that the address value of the next segment,f_{n-1}, should be higher than all addresses of its parent node $Adr(f_n)$ (Equation (3.4)):

$$\begin{cases} \sum_{h=1}^{u} Adr(f_n)[k] \oplus Adr(f_{n-1})[k] = n-1 & (3.3) \\ Binary[Adr(f_{n-1})] >= MaxAdr(f_n) & (3.4) \end{cases}$$

Here, $Adr(f_n)[k]$ and $Adr(f_{n-1})[k]$ are the values of the k^{th} bit of the addresses $Adr(f_n)$ and $Adr(f_{n-1})$ respectively whereas $MaxAdr(f_n)$ is the maximal value of the memory address that the field $\Phi_2(f_n)$ can provide. Binary(*val*) returns the binary value of *val*. The principle of conversion from binary to Gray code is done according to the following two steps:

(i) The bits of the variable *val* are numbered from the left to the right, from 1 to *u*.

(ii) Bit *h* of the variable *val* is 0 if the bits *h* and *h*+1 of the corresponding binary code word are the same, else bit *h* is 1. When *h*+1 = *u*, bit *u* of the binary code word is considered to be 0.

The IP forwarding algorithm then proceeds to fill the required information of the memory cell corresponding to the actual address, $Adr(f_n)$, by setting the corresponding bit of the field Φ_1 to 1 (e.g. to indicate that the segment f_i is now stored in R_{fwd} table).

The algorithm stores then the longest prefix by modifying the fields Φ_3 and Φ_4 according to the following two steps:

(i) Find out the number of bits, *h*, which are defined in *IP_address_Prefix*.

(ii) Set the bit Φ_3[k]: k= 2^{h-1}+ $IP\_address\_Prefix(f_i)$ to one.

(iii) Store the term *nxt_hop* into the Φ_4 field of the R_{fwd} table at the address, $Adr(f_i) = GetValuefromMemory(Adr(f_{i-1}))$. The transition from the variable *val* to *adr* requires then reversing only one bit of *val*.

4. COMPARISON WITH OTHER PREVIOUS WORKS AND THE NEED FOR DYNAMIC RECONFIGURABLE CIRCUIT

In contrast, to the tree-bitmap algorithm reported in [10], our algorithm proceeds to reserve the memory words only to the segments that have been defined during the lookup table construct. This leads to a better utilization of the forwarding table memory, which constitutes an important design factor because of the high per-word cost of the lookup table memories. In addition, because of its hardware complexity, the tree-bitmap algorithm presents a significant computation overhead. In [16], the computation time was found to be $\frac{1}{8}$ slower than the memory bandwidth. The next section will show how the hardware implementation provides a smaller hardware complexity and high performance.

4.1 Our Architectural Model: Dynamic Logic Circuit (DRL)-based Programmable Router.

In the past few years, several types of architectures targeting Internet routers have been reported [7][16]. Field Programmable Gate Array (FPGA)-based routers were particularly attractive because of their capability to be adapted to new networking protocols (e.g. better than ASIC), in addition of providing higher performance than processors. The drawback of static FPGA circuits however is that the maximal throughput they can provide directly depends on the complexity of the application. This is due to the routability problem between the reconfigurable logic blocs. Hence, lot of research efforts are being focusing on new high level synthesis techniques targeting FPGA circuits to allow the designers improving the quality of their design within a short time.

In this paper, a new programmable router architecture using our Dynamic Reconfigurable Logic (DRL) circuit [9][16] is proposed (Fig. 2). The on-fly reconfiguration of the DRL circuit overcomes most of the limitations of static FPGAs.

Modular design techniques were employed throughout the design to provide scalability for various system configurations. It's composed of three modules: the DRL VLSI circuit, control Processor, and Dynamic Reconfigurable Switch (DRS). In out application, the hardware extracts the IP address from incoming packets and writes its value to the input FIFO. Next, the address is read by one of the DRL circuit, which proceeds to perform the IP forwarding algorithm (after running checksum). The packet is then transmitted to the FIFO corresponding to the appropriate next hop. The

next sections will highlight each of these modules in addition of the main advantages of the new architecture for the proposed new algorithm compared to traditional static-FPGA-based routers.

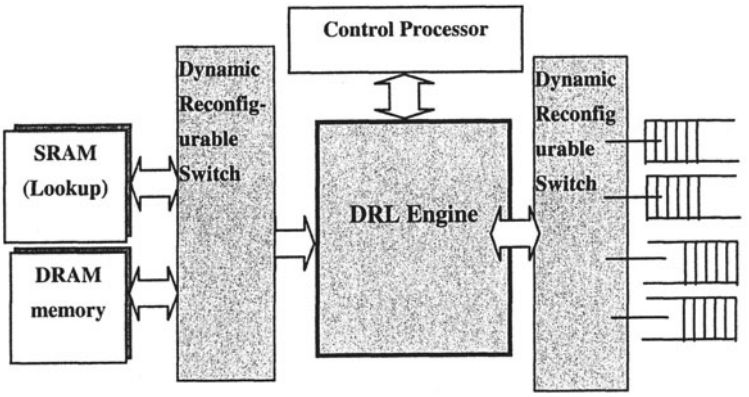

Figure 2. Block Diagram of Router with DRL engine.

4.2 Dynamic Reconfigurable Logic (DRL) Circuit.

The DRL circuit is the core hardware unit of our router (Fig. 3). In order to allow several tasks other than IP forwarding, such as QoS processes, to run concurrently (or in pipeline), a distributed control path organization was designed. Each control path communicates with its independent data path through an independent control and data bus. The control path is composed of a set of registers, a program sequencer, and a program/data memory.

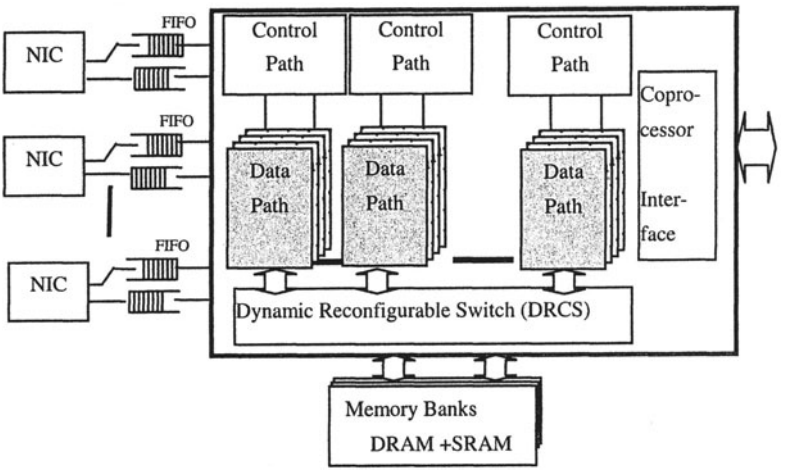

Figure 3. Architecture of the DRL-based Network Processor.

The main task of the Program SEquencer (PSE) module in the control path (Fig 4) is to decode and execute an instruction. Most of the instructions can be executed in one DRL's clock cycle. These instructions can perform **(1)** switch to another context, **(2)** load a register with an immediate value, **(3)** perform a two operands arithmetic/logic operation, or **(4)** conditional/ unconditional branch statements. The control bus connecting the control path to the data path carries the result of the last operation executed by the data path to inform the PSC which portion of the control path or which data path context should run next. The data path hardware is composed of 8 contexts. Each context consists of a two-dimensional matrix of Processing Element (PEs) of 8 inputs/4 outputs each.

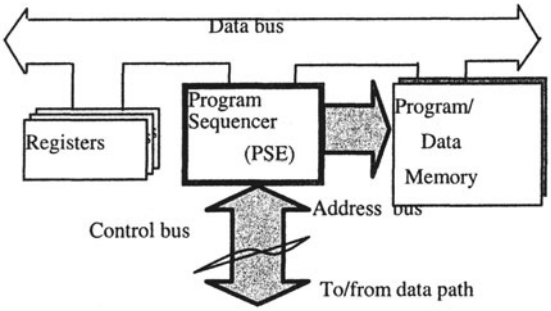

Figure 4. Functional Block Diagram of the Control Path.

Thus each PE can handle any arithmetic/logic operation of 2 input operands of 4 bits each and an output of 4 bits. This leads to croaser granularity of the chip than traditional FPGA circuits such as Xilinx's Virtex chip [11] where their PEs (e.g. configuration logic bloc) can accept at most two bits per operand. Another difference arises from the fact that the operation to implement into the PEs does not use Lookup up table mechanism but is hardwired to allow a faster execution (Fig. 5(a)). Actually, five hardwired operations ("add", "sub", "or", "xor", and "nand"), which are abundant in networking protocols' kernels, are supported. Any other operation can be implemented by combining few of these operations. Each PE contains a 4 bits register, R, for storage and a set of 8 Configuration Bits (CBs) leading to 8 contexts on the circuit (Fig. 5(b)).

This distributed architecture (control/datapath) allows several datapaths to run concurrently without degrading their performance since only the active datapaths are actually mapped onto the datapath hardware. The other portions of the program are stored in an off-chip memory and downloaded into one of the eight contexts of the DRL circuit only one they are reached by the program.

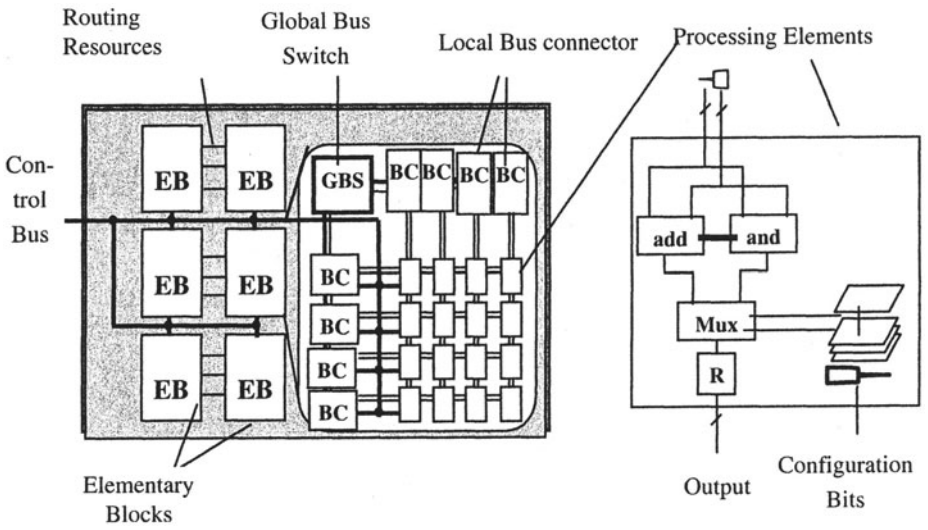

Figure 5. Data Path Architectural Model of our DRL Circuit.

(a) DRL Block Diagram. **(b)** PE Structure and its Interface with the Configuration Bits.

4.3 Control Processor

The main task of the control processor is to download configuration bit streams into the DRL circuit. Other not less important tasks performed by the processor concern management processes such as balancing the workloads among the DRL circuits (in case of several DRLs circuits) and updating the lookup table. These are relatively rare events (even ten-thousand route updates per second consumes less than 1 % of SRAM bandwidth); therefore one single processor is enough.

4.4 Dynamic Reconfigurable Control Switch

The Dynamic Reconfigurable Control Switch is a hardwired switch engine used to dynamically allocate the DRL's IO ports to SRAM and DRAM memory buses, with a very short propagation delay. The use of such high-speed switch is suitable for improving the quality of the FU sharing when the memory or I/O accesses are abundant, which is the case of network protocols. Dynamic allocation of the memory banks to the DRL's ports is one of the tasks performed by our HLS algorithm [9, 12].

4.5 Hardware Implementation of the IP forward Algorithm

Figure 6 shows the hardware implementation of the algorithm into our DRL circuit. The input of the data path is the DATA_IN_REG register. It contains the value of the actual destination IP address, which is split into n registers to store the q segments: F_n_REG to F_1_REG. These registers are fed to an (n:1) multiplexer. Its output, which is controlled by the control path CP1, is the input of a shift register, the output of which indicates if the actual segment has or not a child node. This information is collected by the control path, CP2, which activates either the Get Nxt_hop or GetNxt_Adr modules. Here, our algorithm requires smaller hardware complexity (1 shift register, 1 (n:1) multiplexers, and 2 adders) than the tree-bitmap algorithm, which requires at least 21 adders (among them 15 are in the same critical path) and 4 multiplxers [16]. Additionally, the distributed architecture of our DRL circuit make easier the use of the routing resources between the PEs which lead to short propagation delay between the arithmetic and logic operators. Hence, using the intrinsic features of the DRL, Gbps line speed can be reached.

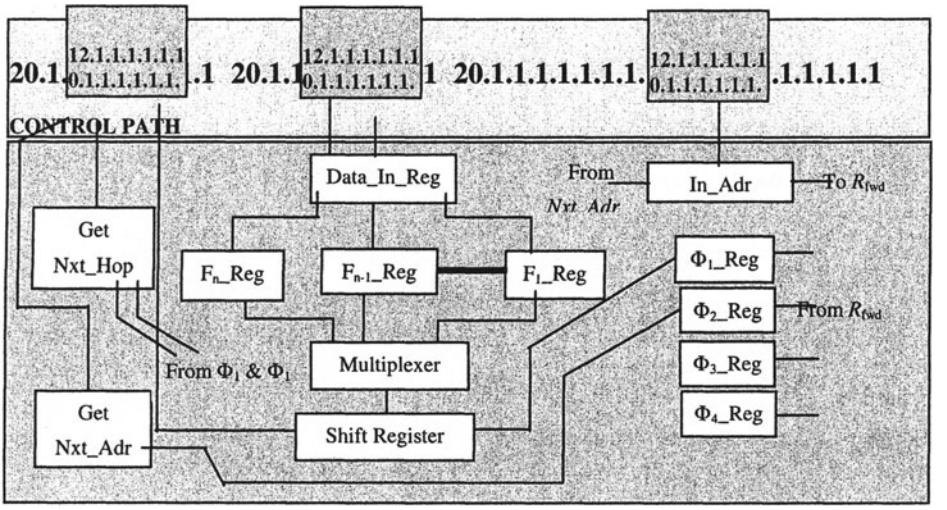

Figure 6. Hardware implementation of the IP forwarding algorithm into our DRL circuit.

5. EXPERIMENTAL RESULTS

To demonstrate the correctness and the performance of our new hardware algorithm, several software development package tools have been explored. The first set of experiments was conducted to test the behavior of the algorithm in a large-scale network and deduce different hardware parameters of the programmable router (e.g. length of the input and output queues, etc...). This has been done using the Ns2 simulator [17]. Ns2 is a discrete event, parallel simulation environment implemented in C++ language. Prior to the experiments, our IP forwarding algorithm and its corresponding lookup update algorithm were first compiled (The total size of resulting object files was equal to 34 Kbytes) and linked with the Ns2 libraries. Finally, all simulations used TCP transport layer protocol. The network infrastructure used for this purpose is composed of a daisy-chain of 30 Internet routers connected with each other through links of 1 Mbps bandwidth and 1 ms delay (Figure 7). Each Internet router was connected to 90 nodes. The node, $n_1(R_1)$ connected in the most left router then keeps sending its packets (64 bytes each) to consecutively three nodes connected to the most right routers ($n_1(R_{30})$, $n_2(R_{30})$, and $n_3(R_{30})$) after 1 second, 5 seconds and 20 seconds respectively. At time t = 25 seconds, the sender node $n_1(R_1)$ stop sending the packets. The simulation last at time t = 40 seconds. The received packets, along with their time stamp are stored in the files out_1, out_2, and out_3 respectively.

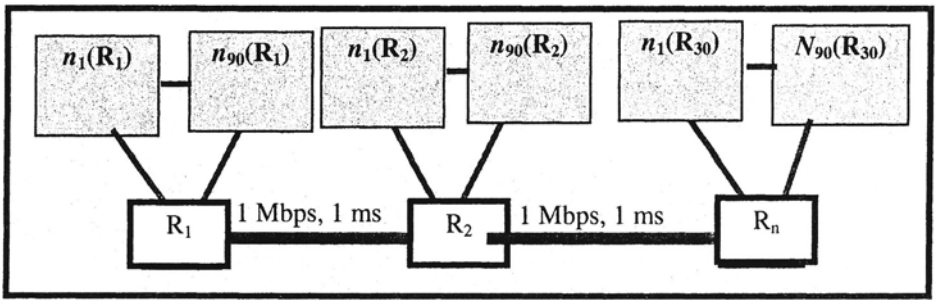

Figure 7. Simulation Topology using Ns2 in order to test the correctness of the algorithm.

The first set of experiments was conducted to observe the correctness and behaviour of the algorithm function of the size of the queue of the routers. Figure 8 shows the results of the stream out_1 after 1 second, 6 seconds and 23 seconds. We can observe that the size of the queue is significant only for 6 and 23 seconds because of the congestion in the

network on both these times. Nevertheless, the correctness of the algorithm is verified since all the packets were received by the three receivers at the time 40 seconds.

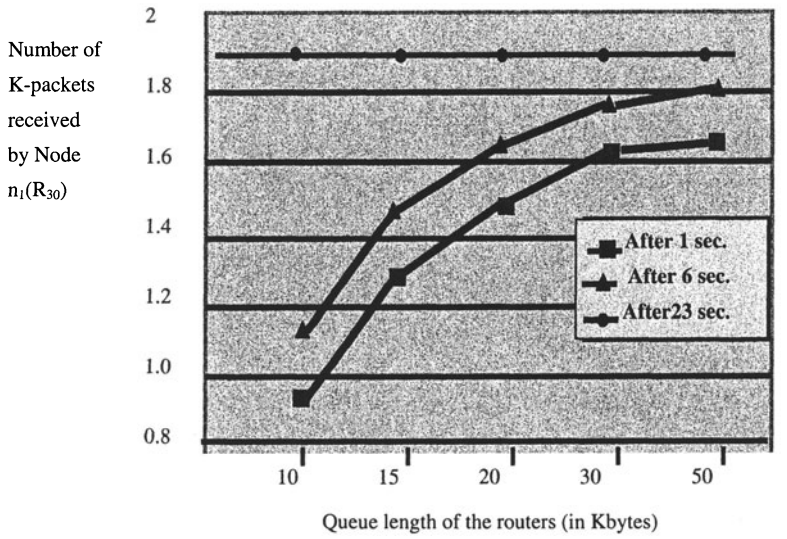

Figure 8. The number of packets lost function of the size of the queue.

To test the speed-up of our hardware algorithm, a hardware implementation of the programmable router was performed using Verilog HDL language and synthesized using Mentors Graphics' software [18]. It required 2400 lines and 320 of VHDL lines code for the DRL circuit land the control processor respectively. The segment's size of the destination IP address was set to 4 bits. Therefore, each memory word of the R_{fwd} requires 58 bits. This leads to a total SRAM capacity of 512 KB. The SRAM memory bandwidth was equal to 10 *ns* whereas the total capacity of the DRAM memory to store the entire packet (i.e. header and payload) was fixed to 32 MB. The clock speed of the DRL circuit was set to 180 MHz.

To investigate the performance of our hardware algorithm on our DRL circuit, a number of IP routing tables were collected. Internet routing tables are currently available at the web site for the Internet Performance Measurement and Analysis (IPMA) project [13] and were previously made available by the now terminal Routing Arbiter project [14]. The collected routing tables are daily snapshots for the routing tables used at various large Internet interconnection points. Some of the routing entries in these tables contain multiple next hops. In that case, one of them was randomly selected as the next hop to use in the forwarding table. Using a portion of the Ma-

West snapshot from July 2001, a data structure consisting of 16,564 routes was loaded into the SRAM memory. Our lookup update hardware algorithm generated 15564 routes, whereas the tree-bitmap data structure occupied 55,158 words of memory. Alternatively, the DRAM memory stores different IP packets. In addition, 2048 IP packets were randomly selected from the table snapshot.

The DRL engine was configured to instantiate multiple threads in order to scale the lookup throughput with systems demand. Test runs were initiated using configurations 1 to 8 context in the DRL circuits. Figure 9 shows the results of the test runs of the worst case tree-bitmap-based IP forwarding alone without intervening the update traffic. The results were compared with those reported in [10], which use the Xilinx's Virtex FPGA [11]. The worst-case performance is reached when the actual bit to check is in the last position of the external bitmap field (i.e. eighth bit position), which corresponds to a chain of 15 adders. The implementation of the tree-bitmap algorithm into our DRL circuit results in a delay of 15×2 ns (= 30 ns) for calculating the intermediary results. Thus, at the most-critical case one packet will be treated in $24 \times \frac{30}{8} ns$ = 90 ns (30 ns was divided by 8 because 8 contexts can run in pipeline). This is equivalent to 11,12 Mpackets per second throughput of IP packets, which corresponds to a speed of 5,56 Gbits speeds for packets of minimal size of 64 bytes each, which is an improvement over the Lucent's HSSI C Series IP Forwarding router which provides 900,0000 packets per second [13].

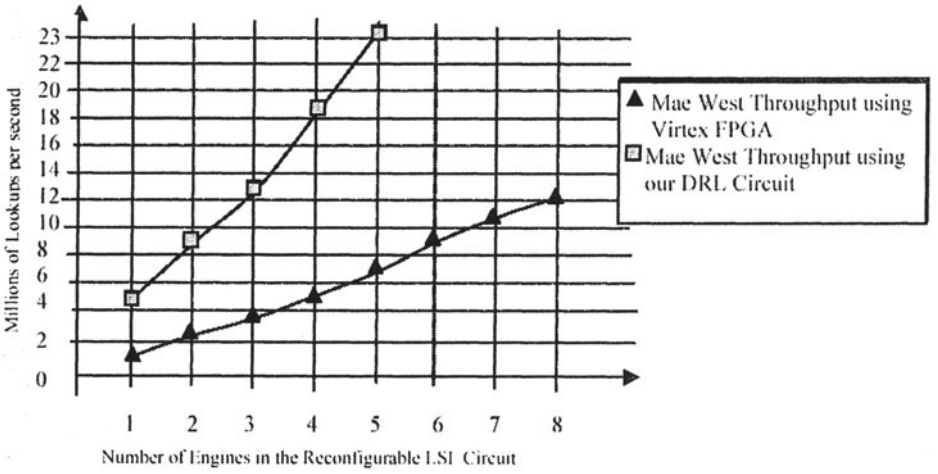

Figure 9. Results of IP forwarding on Mae-West Snapshot.

In the case of our hardware IP forwarding algorithm, the total delay to calculate one intermediary address was equal to 5 ns, which corresponds to a latency of $24 \times \frac{5}{8} = 15$ ns per packet. Therefore, a total speed of 66,67 Mpackets per second which corresponds to a speed of 33,34 Gbps line speed is possible using a single DRL circuit. Furthermore, since the new algorithmuses only a small fraction of processing elements of the DRL circuit, then other QoS algorithms can be easily supported.

6. CONCLUSION

In this paper, a new hardware algorithm for IP forwarding is presented. It features low computation time overhead and efficient use of the Lookup-table memory. In addition, a dedicated new DRL-based programmable router architecture is designed and implemented. Compared to traditional reconfigurable circuits, our DRL circuit features a distributed architecture of the control path, which has the advantage to enhance the pipelining/parallelism, and facilitates the Hardware synthesis algorithm, by separating the synthesis of the control and data paths. However, in traditional static FPGA circuits, the control path uses slow logic blocks. The other feature of our architecture is that unlike traditional reconfigurable circuits, our DRL circuit uses as basic processing element not Lookup tables, but hardwired functions. This has the merit to increase the overall speed of the data path.

Experimental results show that using one DRL circuit, a performance of 33,34 Gbps (OC576) can be reached using a small number of processing elements of the DRL circuit. Therefore, our proposal can be considered as potential candidate for next generation router architectures.

REFERENCES

[1] Internet Routing Table Statistics, "http://www.merit.edu/ipma/outing_table/, May 2001".
[2] S. Fuler, T. Li, and K. Varadhan, "Classless inter-domain routing (CIDR): An address and aggregation strategy", RFC 1519, September 1993.
[3] Pankaj Gupta. Steven Lin, and Nick McKeownm "Routing Tables in hardware at memory access speeds", in IEEE Infocom, 1998.
[4] S. Nilson and G. Carlson, "Fast IP lookups for internet routers", in IFIP International Conference of Broadband Communications, 1998.
[5] V. Srinivisan and V. Varghese, "Faster IP lookups using controlled prefix expansion", in SIGMETRICS, 1998.
[6] M. Degermark, A. Brodnick, and S. Pink, "Small forwarding tables for fast routing lookups", in ACM Sigcomm, 1997.
[7] SiberCode Technologies Inc., "SiberCAM Ultra-2M SCT200", Product Brief, 2000.

[8] Marcel Waldvogel, Georges Varghese, John Turner, and Bernhardd Plattner, "Scalable high speed IP routing table lookups", in Proc. Of ACM SIGCOMM'97, pp. 25-36.
[9] M. Meribout and M. Motomura, "Method for compiling high level programs into hardware", *Japanese Patent*: JSP2000-313818, 2000.
[10] W. N. Eartherton, "Hardware-based Protocol Prefix Lookups", thesis, Washington University in St Louis, 1998.
[11] "The Viretex user data sheet", Xilinx, 2001.
[12] M. Motomura et al, "An Embedded DRAM-FPGA Chip with Instantaneous Logic reconfiguration", *Symposium on VLSI Circuits*, pp. 55-56, July 1997.
[13] Michigan University and Merit Network. Internet Performance Management and Analysis (IPMA) Project. Details available at http://nic.merit.edu/~ipma.
[14] The Routing Arbiter Project. Internet routing and network statistics, http://www.ra.net/statistic.
[15] M. Meribout and M. Motomura, *New Design Methodology with Efficient Prediction of Quality Metrics for Logic Level Design Towards Dynamic Reconfigurable Logic*, International Journal for Systems Architecture (JSA), (Elsevier Science Ltd), vol. 48/8-10, pp. 285 – 310, 2003.
[16] David E. Taylor, Jon Tunner, and John W. Loclwood, "Dynamic Hardware Plugins (DHP): Exploiting reconfigurable hardware for high performance programmable routers," In IEEE OPENARCH 2001, Anchorage, AK, April, 2001.
[17] *ns* notes and documentation. The VINT Project, UC Berkeley, LBL, USC/ISI, and Xerox PARC, November 1997. Available from http://www-mash.cs.berkeley.edu/ns/.
[18] URL: www.Mentorgraphics.com.