Design of a Web Switch in a Reconfigurable Platform

Christoforos Kachris, Stamatis Vassiliadis
Department of Electrical Engineering,
Mathematics and Computer Science
Delft University of Technology
The Netherlands
{ kachris, stamatis } @ce.et.tudelft.nl

ABSTRACT

The increase of the web traffic has created the need for web switches that are able to balance the traffic to the server farms based on their contents (e.g. layer 7 switching). In this paper we present a web switch implemented in a multi-processor reconfigurable platform augmented with hardware co-processors. The system supports the TCP splicing scheme to accelerate the routing of the packets by forwarding packets at the IP layer after a connection has been spliced. The processors are alleviated using special co-processors for the management of the spliced connection and the URL string parsing. The proposed scheme can sustain up to 927Mbps throughput for 64KB request file size consuming less than 1Watt in a Xilinx Virtex4 FPGA. Hence, the system provides an efficient combination of processor's flexibility and ASIC's performance. Finally, the system is compared against a network processor-based and a software content-based switch in terms of performance, area, and power.

Categories and Subject Descriptors

C.2.1 Network Architecture and Design

General Terms

Performance, Design

Keywords

Reconfigurable logic, web switch

1. INTRODUCTION

As the web network traffic keeps increasing there is the need for the ISPs and search engines to accommodate server clusters to face the increased demand. The most straightforward way to balance such a cluster is to use a switch that distribute the network workload based on layer 3 and layer 4 information (such as the IP address and the TCP or UDP port). However, the load balancing based on these layers is not efficient enough. First of all, the data must be replicated to each server; hence any change in the server content need to be updated on every server. Furthermore, the use of NAT proxies results to same IP address for a large number of clients. Hence, the use of IP address to balance the system is not efficient. Moreover, the network balancing, using layer 3 and layer 4 information, can not face more sophisticated requirements, such as content customization (e.g. content based on language or geographic region). The solution is the switching based on layer 7 information. Layer 7 load balancing provides 3 major advantages according to Cisco [1]:

- *Scalability and acceleration of the application*
- *Persistent user sessions on a server*
- *Content customization based on user profile*

Layer 7 load balancing based on the URL (Uniform Resource Locator) string can be performed either by distributing the data based on the directory domain or based on the file type (e.g. html, image, video, script). Load balancing based on the URL can also provide faster web response (download time) if specialized servers are used for each application. Furthermore, layer 7 load balancing provides session persistence. Session persistence can be used to forward the traffic of one client always to the same server in which previous data (transactions) are stored. Session persistence can be based on an HTTP cookie, a URL string or an SSL session. Finally, content-based switching can also provide content customization based on language or geographical region.

Web switches were initially implemented in software targeting general purpose computers. The main drawback of using a software web switch for web balancing is the additional latency that is introduced. After a web switch has identified the corresponding server, the packets have to travel through the web switch to change the packet's header. The latency is mainly introduced because of the protocol stack processing. The packets have to travel until the upper layer where the switch module will process and forward the data. But after a connection has been established between a client and a server, the processing requirements are usually some header modifications. Hence, the TCP splicing scheme was introduced in 1998 [5]. The TCP splicing is used to forward the packets at the IP layer (layer 3) instead of the upper layers, after a connection has been established. Hence, the latency is reduced significantly and the sustained throughput is increased.

This paper presents a software/hardware co-designed content-based (web) switch which has been implemented in a multi-processor reconfigurable logic platform (Xilinx Virtex 4 FPGA). The main software part is executed by the Xilinx's MicroBlaze 32-bit customized soft-core processors, while the most demanding functions such as the URL parsing has been implemented in hardware co-processors. Furthermore, the system supports TCP splicing. The main advantage of the web-switch implemented in reconfigurable logic is two-fold. First of all, the application is programmed in a customized RISC processor in C instead of vendor specific assembly language. In addition, the proposed scheme can be extended with additional modules that can be used for other operating modes (such as cookies-based switching, payload-based switching) or additional payload processing (encryption, compression, intrusion detection). These operations would be too time-consuming to be performed by the current network processors.

The main contribution of this paper is:

- The implementation of a web switch in a multi-processor reconfigurable platform supporting TCP splicing
- The proposed Connection Manager and the URL string parsing co-processors
- The performance, power, and area evaluation and the comparison against a network processor-based and a software-based scheme

This paper is organized in the following way. Section 2 presents the related work in terms of content-based switching. Section 3 presents the organization of the proposed scheme, the customized processors and the co-processors. Section 4 presents the evaluation of the system in terms of performance, power, and area and the comparison against other schemes. Finally, section 5 presents the conclusions of this work.

2. RELATED WORK

Until now, three schemes have been proposed for the implementation of a content based switch. A software approach (using a general purpose processor) provides great flexibility but has limited performance. A hardware approach (ASIC) has increased performance but lacks of flexibility. The most recent scheme is the use of network processor that combines the flexibility of processors with the throughput of the ASIC using basic hardware coprocessors. However, these coprocessors are usually modules that are used in general packet processing (CRC, header checksum, etc.) and can not be used for dedicated functions such as payload processing or URL parsing. Furthermore, the network processor platforms incorporate multiple instances of specialized processors that are programmed using vendor specific language (e.g. the microcode used by the micro-engines processors in the Intel's IXP network processors).

In [8] a content-based switch is presented, that has been implemented in hardware. The system consists of input and output controllers to process the data packets, while the parsing of HTTT packets and URL based routing is performed by a general purpose processor. This scheme can support up to 500 ops/sec with 50msec latency but it can not be extended or modified in the data plane path.

In [2], [3] a content based switch has been proposed targeting the Intel IXP network processor. The IXP network processor consists of 8 multi-threaded micro-engines and each micro-engine is able to support up to 8 threads. The application has been partitioned into 4 micro-engines. Two of them are used for the input and output packet processing, one is used for the client's packet processing, and the last one for the server's packet processing. The proposed system supports also TCP splicing to accelerated the throughput and reduce the latency. Furthermore, a similar design implemented in software (Linux) has been used for comparison. According to this paper the network processor scheme can support up to 700Mbps throughput for 1 MByte of requested file size, while the Linux scheme can support up to 320 Mbps for the same requested file size. Using 1 Kbyte requested file size the throughput is 8.2 and 46.6 Mbps for the Linux-based and the network processor-based scheme, respectively.

In [4], a similar scheme is proposed based on Intel network processors but in this case the data processing is performed by the micro-engines while the control processing is performed by a host processor. Furthermore, the general purpose Strong ARM is used to control the micro-engines. This scheme is able to achieve 3.47 Mpps (Million packet per seconds) using the older IXP 1200 network processor.

In [5], the TCP splicing mechanism is proposed and a software implementation under the BSD operating system (BSDI BSD/OS 3.0) is performed targeting a general purpose processor. The proposed implementation is able to sustain up to 70Mbps (in 1998) with almost 85% CPU utilization. The mean forwarding latency was 102 ms while the latency of a simple forwarding scheme is 92 ms.

In [6], there is one more implementation of the TCP splicing as a Linux kernel module. In this case, the switching is performed based on the URL and then the connection is spliced between the clients and the servers. The system was based on Fast Ethernet Network Interfaces (100Mbps) and the URL-aware switch was based on a Pentium III 555MHz CPU. According to the paper, the system was able to issue 233K connection with 1KB file size with 61.32% CPU utilization.

In the area of string matching there are many papers that propose several hardware implementations ([18][19]). The most efficient ones can be implemented in reconfigurable logic (FPGAs, [20]) but the main disadvantage is that each time that the string patterns change, a new reconfiguration is required. The time to synthesize, place and route the new design and reconfigure the device is prohibited for applications such as firewalls or web-switched. In these applications the user must be able to apply in short time the new rules for string matching.

In this paper we propose a scheme based on reconfigurable logic (FPGA) using 32-bits customized RISC processors with specialized hardware co-processors for content-based switching. The co-processor that is used for URL switching is based on a scheme in which the data are stored in internal RAM, hence it can be easily updated without reconfiguration. The processors can be programmed using C and not vendor specific language. Hence, we claim we achieve both the required flexibility and increased performance. Furthermore, the design can be extended with additional co-processors for specialized payload processing (such as encryption, compression, or intrusion detection). The main concern of the FPGAs are the increased power consumption compared to an ASIC, hence a power evaluation is performed to compare it with other schemes.

3. ORGANIZATION

The main goal of the proposed scheme is to provide the flexibility of the network processors combined with the increased performance of an ASIC keeping the power consumption low. Hence, a careful software-hardware partitioning has been performed. The system level organization is shown in Figure 1. The system consists of two 32-bits RISC soft processors (each one with a 16KB separate instruction and data memory) and a number of co-processors used to accelerate the performance of the system.

As the processing requirements of the network traffic increase, many network processor vendors have used multi-processors platforms to face the processing demand. There are two methodologies to exploit the additional processing power. Either to connect the processors to a pipelined scheme in which the output of one's processor is the input to the next, or to connect them in parallel in which the whole packet's processing is performed by each processor. The main drawback of the first solution is that a careful partitioning must be performed to each pipeline stage to avoid an unbalanced system. Hence, each change in the application would require a different partitioning. Thus, we have used the parallel scheme in our design in which the program code is the same to both processors and there is no need for partitioning.

The processors, the co-processors and the memory units are all connected using a 32-bits common bus (OPB) which supports burst transactions. Two Dual-Port Block RAM modules (16KB BRAMs) have been used for the buffering of the client's and server's packets attached to the OPB bus. The other port of the memories is connected to the Xilinx's

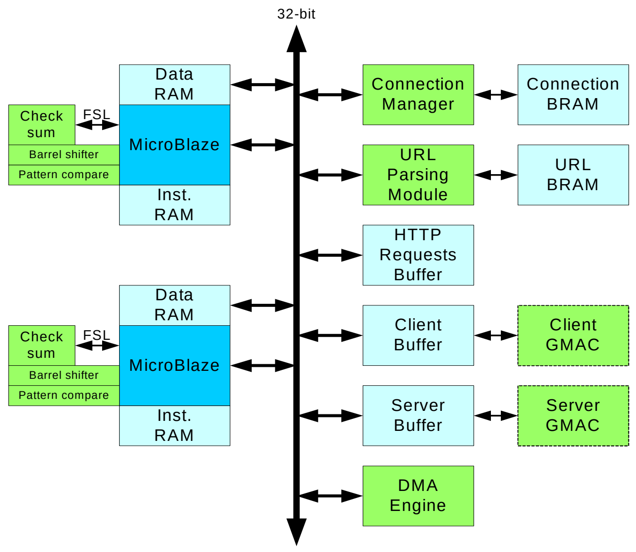


Figure 1. System level architecture

Gigabit Media Access Controller (GMAC). The first GMAC is connected to the client link while the second one is connected to the server farm. Furthermore, a 32KB BRAM is used to store the HTTP requests.

The most time consuming functions of the web switching application are the management of the spliced connections and the URL parsing. To accelerate the performance of the system, these two functions have been implemented in hardware and are connected to the processors using the shared OPB bus. The Connection Manager Module is used to control the spliced connection table (the RAM in which the connections are stored). The URL Parsing Module is used as a search engine. The input is the URL string and the output is the corresponding IP address of the server. Every block RAM (BRAM) is on-chip (internal) for faster access but the system can be easily extended to use external RAM to support more connections, causing some reduction in performance.

3.1 The MicroBlaze processors

The processors that have been used are the Xilinx's MicroBlaze processors [14]. Theses processors have been customized to include a barrel shifter and a pattern compare unit. The barrel shifter instructions takes 2 cycles while the pattern compare instructions (such as pattern compare equal) takes 1 cycle. The use of these additional units provides improved performance (40%-50% fewer cycles compared to the initial configuration) since the majority of the network processing application use bit-wise instructions, while the area overhead is minimal. Furthermore, the processors have been extended using a checksum module for the packet's header processing. The checksum module is attached to the processor using a point-to-point interface called FSL (Fast Simplex Link [15]) that provide a fast interface between the processor and the co-processor.

The application that is hosted in the processors is a simplified version of the TCP splicing used in Linux [10] that supports the most important features of the TCP splicing. The simplified version of this application has the same characteristics as in [3] for the TCP splicing targeting the Intel network processor. Each processor polls the Client and the Server buffer for new packets. If there is a new packet, only the header of this packet is forwarded to the processor. The processor firstly verifies the IP and the TCP header using the checksum module. If a packet arrives with the SYN flag on, the processor sends a command to the Connection Module to add the new connection. After the processing of the packet, if there is a new packet created, then the packet is forwarded to the corresponding buffer (client or server). When a new http request arrives, the processor stores the packet in the Request BRAM. After a connection has been established between the content switch and the server, the Request packet is retrieved from this BRAM and is forwarded to the server buffer. Moreover, after the connection has been established, the packets are forwarded from the client buffer to the server buffer and vice versa using DMA burst transfers. Hence, in this case the processors are used only to check the established connections and change the packet's header (such as the IP, the TCP Sequence number and the header's checksum).

A list of free indexes is stored also in the shared memory that stores the request packets. Hence, every time a processor wants to store a request, firstly it requests an available index from the free list table and then stores the request to the provided index. In order not to have deadlocks, a test and set mechanism is used for the shared memory. The processor first locks the shared memory and after receiving the required index it unlocks the memory. Hence, the consistency is preserved between the two processors.

Moreover, in order to hide the communication overhead between the processor and the buffers, the following scheme is used. The Data BRAM is a dual port RAM, in which one port is connected directly to the processor while the other port in connected to the OPB bus using a bus controller, which supports DMA burst transfers. Hence, while a packet is processed by the MicroBlaze the next packet is loaded to a reserved space in the Data BRAM using DMA transfers. Thus, the communication overhead is hidden by the packet processing.

3.2 The Connection Manager

In order to accelerate the processing of the packet, a co-processor has been created to manage the connection table. The Connection Table is organized as it is shown in Figure 2. Each entry consists of the Client's IP address, the Client's TCP port, the Server's IP (SIP) in which the connection has been spliced, the state of the connection (idle, connected, spliced, etc.) and the initial TCP sequence number (SEQ field) of the client and the server. The Server's IP is 8 bits (the last 8-bits of the 32-bit IP address, since the first 24 bits of the IP address are usually the same for a server cluster), thus it can support a cluster server of up to 256 servers, which is adequate in most of the cases. The current implementation can support up to 4K connections while it can be easily extended to support more connections. The Connection Module supports three commands:

- ***Write*** *(IP, TCP, SIP, state, Client SEQ, Server SEQ) return 0;*
- ***Search*** *(IP, TCP) return SIP, state, Client SEQ, Server SEQ;*
- ***Delete*** *(IP, TCP) return 0;*

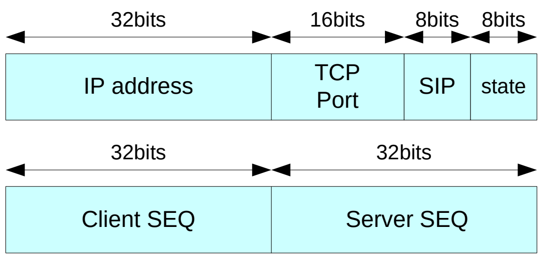


Figure 2. The connection entries

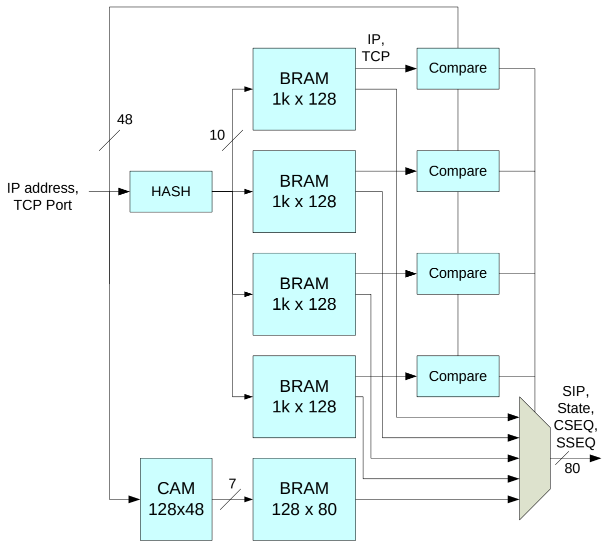


Figure 3. The Connection Manager

The organization of the Connection Manager is shown in Figure 3. A hash unit is used to create a 10-bit index out of the 32-bits IP and the 16-bits TCP input. When a new entry command is sent, the Connection Manager tries to store the entry into one of the Block RAMs. In case that all of the BRAMs are occupied (collision) a small CAM (Content Addressable Memory) is used. The CAM stores the IP and the TCP field (48-bits) and outputs an index to a RAM in which the remaining data are stored (Server's IP, Connection's state, etc.). When a search is performed, the

Connection Manager searches each of the BRAMs and if there is a match in any of them it outputs the result. In case that the connection data has been stored to the CAM then the multiplexer selects the CAM output. The CAM that we used takes 1 clock cycle for read and 16 clock cycles to write. As is it shown in [11], using four block RAMs we can reduce significantly the probability of collision. This scheme can provide a result of a search only in 4 cycles.

To measure the efficiency of the Connection Manager we used the UC Berkeley Home IP Web Traces [17]. After extracting the IP address and the Port number of the clients, we measured the number of BRAM collision for 4096 connections using a simple XOR hash function. The number of collisions (that hash to the same address over four times) was 295. The CAM can accommodate 128 entries, hence the probability of a connection collision (in which the connection can not be stored neither at the BRAM nor at the CAM) is 4%. Table 1 shows the probability of a connection collision for several numbers of BRAMs and for 128 and 256 entries CAM. This scheme can be easily extended using larger FPGA device. In case that we use four 16Kx128 BRAMs and a 512 entries CAM then the probability of connection collision is 5% (there were 1383 BRAM collisions) and is able to support 64K connections.

Table 1. Collision Probability

Number of BRAMs	Collision Probability	
	128CAM	256CAM
2	18%	15%
3	11%	8%
4	4%	1%
5	0%	0%

3.3 The URL parsing coprocessor

One of the most computational demanding function, is the web switching based on the URL string. For example content distribution can be performed based on the URL (e.g. the *www.foo.edu/publications* requests could be distributed to a separate server than those to www.*foo.edu/people*). Another option is the switching based on the application. For example the HTTP request for an image could be forwarded to a separate server than the server for the HTML pages. Hence, an efficient URL parsing module is necessary to perform fast web switching based on the URL. In our implementation an efficient form of a compresses trie has been used. Three separate block RAMs are incorporated; a 256x32 bits Char RAM, a 256x32 bits Address RAM and a 256x12 bits Control RAM as it is shown in Figure 4 . The Char Table stores the characters of the URL string, the Address Table stores the Server's IP and an index to the Char Table, while the Control Table stores several control bits. After a search has been performed, the Server's IP (the last 8-bits) is stored to the Connection Table (Figure 2). To explain the function of the URL module an indicative example is used, as it is shown in Table 2, for URL directory based switching. This table shows the URL and the last byte of the IP address (the first three bytes are the same for a server cluster) of the corresponding server. As it is shown, we need a co-processor that will be able to perform longer prefix matching of the URL string to the URL table.

Table 2. Directory-based URL Table

URL	Server IP
/pub/*	16
/pub/phd/*	17
/pub/msc/*	18
/people/*	19

The characters are stored in the Char Table using the following algorithm. As long as the char are unique compared to the other characters in the same position, the characters are stored in the Char Table sequentially. When there are different characters in the same position (e.g. p**u**b and p**e**ople) these characters are stored in a separate entry (e.g. address 2, Figure 5) with an index to the Address Table for the next characters (e.g. address 3 and address 7 for *pub* and *people*, respectively). The *L* and the *S* columns of the Control Table are used to indicate if the corresponding entry has characters that split to new sub-string or belong to the same string, respectively. For example, entry #7 stores characters that belong to the same string (hence the corresponding S entry is '1') while entry #4 stores characters that belong to different sub-strings (misc, "phd", and "msc"; hence the L entry is '1'). The C column in the Control Table shows if the string is continued to the next sequentially entry. For example, the "*people*" string is continued to entry #8.

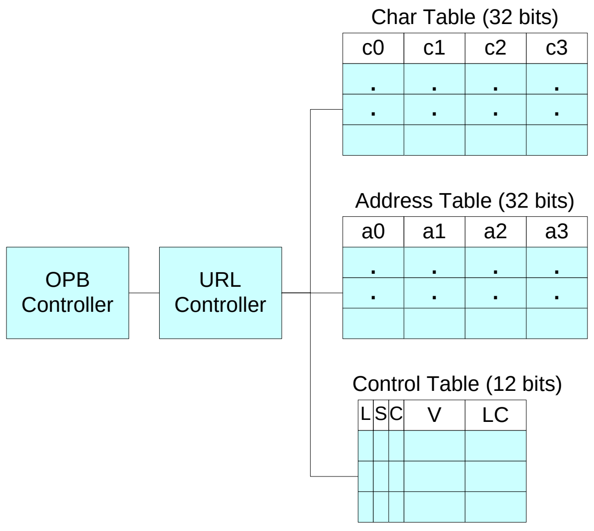
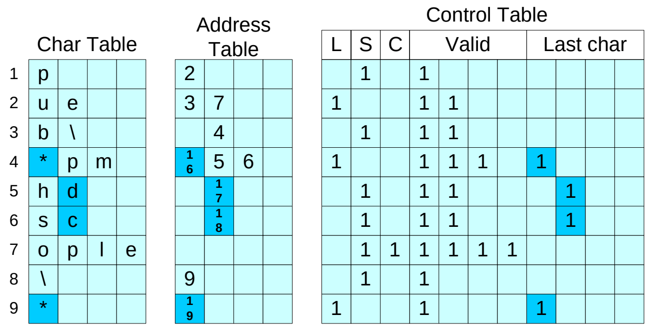


Figure 4. The URL Module



	Char Table				Address Table				Control Table										
									L	S	C	Valid				Last char			
1	p				2					1		1							
2	u	e			3	7			1			1	1						
3	b	\				4				1		1	1						
4	*	p	m		16	5	6		1			1	1	1		1			
5	h	d				17				1		1	1				1		
6	s	c				18				1		1	1				1		
7	o	p	l	e						1	1	1	1	1	1				
8	\				9					1		1							
9	*				19				1			1				1			

Figure 5. The URL Tables

The Control Table holds two more 4-bit columns. The first column represent if a char is valid in the Char Table, while the second column (Last char) represent if this is the last characters, hence there is a valid Server's IP address in the Address Table. For example, in the entries in which the Last Char flag is '1' (the shadow boxes), there is the Server's IP address as it shown in the corresponding URL Table (Table 2).

The URL search algorithm which has been implemented in the URL controller is shown in Figure 6. The controller first checks the same level and the same string flags to identify if the row that is processing holds sequential characters or characters that point to new sub string. In the case that the characters are at the same level (used to point new sub-strings) each character is compared against the current URL char. If it is the same then the next address is retrieved from the Address Table. In the case that the same-string flag is on (second half of the algorithm) the module compared the valid characters of this row with the corresponding URL string.

Using this scheme we can achieve a throughput of 1-4 matched characters per cycle. In order to compute the number of characters that should be stored in each entry of a LookUp table such as the one in Figure 5 we used the http request traces from the San Diego Super Computer Center [12]. According to these traces the maximum number of characters that are used to describe the first two sub-string of a URL string (e.g. \pub\phd) is 27 characters. Taking into account, that the majority of the web-switching is performed using the first two sub-string we set 28 characters the maximum number of characters that can be searched by the URL module. Thus, the maximum number of cycles to output a valid IP address is 28 cycles. The number of entries is expandable and can accommodate a large number of URL string. In the current implementation we used a 1024 entries table; hence the maximum number of stored characters is 4096. However, in cases that the URL entries have higher demands, the URL module can be used as a cache in which the most current entries are stored, while the remaining entries are used by a software function in a separate memory. In addition, the most important feature of this accelerator is that in order to maintain and to update the URL table there is no need for a heavyweight reconfiguration. The URL table is stored in BRAM that can be updated using any external interface (such as UART) and exploiting the second spare port of the inherent Dual Port BRAMs.

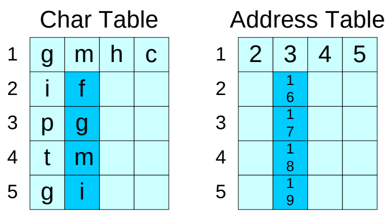
```
% Same Level
if Same_level='1' then
  if char(0)=url(j) and valid='1'
    if Last_Char='1'
      return ServerIP
    else
      address = url(address);
      j = j + 1;
    end if;
  elseif char(1)=url(j) and valid='1'
    …
  elseif char(2)=url(j) and valid='1'
    …
  elseif char(3)=url(j) and valid='1'
    …
  else
    if url_continued='1'
      address = address + 1;
    end if;
% Same String
elseif Same_string='1'
  if c(0)=url(j) & c(1)=url(j+1) &
     c(2)=url(j+2) & c(3)=url(j+3)
    if url_continued='1'
      address = address + 1;
      j = j + 4;
    else if Last_Char='1'
      return ServerIP;
    else
      address = url(address);
      j = j + 4;
    end if;
  elsif c(0)=url(j) & c(1)=url(j+1) &
        c(2)=url(j+2)
    …
  elsif c(0)=url(j) & c(1)=url(j+1)
    …
  elsif c(0)=url(j)
    …
  end if;
```

Figure 6. The URL search algorithm

Furthermore, the URL co-processor can be used for URL application-based switching. In this case, the server is selected based on the extension of the requested file, as it is shown in Table 3. In this example, the requested images (.gif), the videos (.mpg), the HMTL and the CGI files are all forwarded to a separate server. In this case, the processor does not send the first 28 characters. The processor reads the URL characters until a space is found, which means that the URL string is finished. Hence, the last three characters are sent to the URL module for parsing. The corresponding tables for the URL module are shown in Figure 7. The first letter of each application extension is shown in the first row, while the remaining characters of each row are shown in row 2-5. Using this scheme, we can find the corresponding server in N+2 cycles where N is the length of the URL string divided by 4 (the cycles to find the last 4 characters).

Table 3. Application URL Table

URL	Server IP
.gif	16
.mpg	17
.htm	18
.cgi	19


Char Table

1	g	m	h	c
2	i	f		
3	p	g		
4	t	m		
5	g	i		

Address Table

1	2	3	4	5
2		16		
3		17		
4		18		
5		19		

Figure 7. The application based Tables

4. EVALUATION

The system has been implemented into the Xilinx Virtex 4 XCV4LX60 FPGA. The FPGA have usually higher power consumption than the ASICs and lower maximum frequency. On the other hand, the flexibility of the design that they offer is the main advantage. In this section we analyze the performance, the area and the power dissipation of the proposed scheme in order to compare it against a network processor. The main advantage of the current network processors is that they offer a multi-threaded multi-core platform that operates at a high frequency. For example, the Intel IXP 2400 processor provides eight 8-way multithreaded micro-engines that operate at 600MHz. The main drawback is that these micro-engines are programmed into a special assembly language thus it does not provide the flexibility of the common processors. On the other hand, the proposed scheme uses the MicroBlaze processors (operated in 100MHZ) that can be programmed in C, while the most demanding functions (URL parsing, connection Lookup tables) are implemented in hardware that can be reconfigured

4.1 Performance

Both the MicroBlaze's and the miscellaneous modules operate at 100MHz. Two designs have been used for performance evaluation. The first one uses only one processor, hence the overhead of consistency on multi-processors is eliminated (set and lock, etc). In the second case, two processors are used performing the same tasks. The application that it is hosted in the processors is the spliced web switching using the traces from [17] for the IP address and some of the traces from [12] for the URL matching. The requested file size was synthetic in order to measure how the performance depends on the average packet size. Figure 8 shows the performance of the system in both cases for several requested file sizes. The request packet size (the HTTP requested file) is a crucial factor. After a connection has been established, the packets are just forwarded after some of the header fields have been changed. The performance is measured in traffic bandwidth (sustained incoming traffic processing).

Figure 9 shows the utilization of the common bus in both configurations. The OPB common bus is 32-bit bus that operates at 100MHz. Thus, the maximum sustained throughput is 3.2Gbps, but the typical data rate is 1.3Gbps. As it is shown from these figures, in the first case the bottleneck of the design is the processing power of the processor. On the other hand, in the second case, the bottleneck of the design is the bandwidth of the common bus. The network-processor based scheme, published in [3], sustain almost 610Mbps for a 64KB packet size while in the case that the requested packet size is 1Kb the sustained throughput is lower than 100Mbps. Hence, the proposed scheme in reconfigurable platform offers significant higher throughput when the requested file size is small while in the case that the requested file size is larger (16KB) the sustained throughput is almost 55% percent higher. Since, the main bottleneck of the design is the bandwidth of the common bus, for larger requested file sizes (1024KB) the throughput of the proposed scheme drops to almost 20% higher than the network-based scheme.

Table 4 shows the average processing latency for various packet types compared to the network processor-based and the software-based scheme presented in [3]. As it is shown the latency is very close to the scheme based on the Intel IXP network processor. However, this is the latency for each processor. Since two of these processors are used the performance of the system in terms of throughput is almost 55% higher. Furthermore, this table shows that the proposed architecture have similar performance in terms of

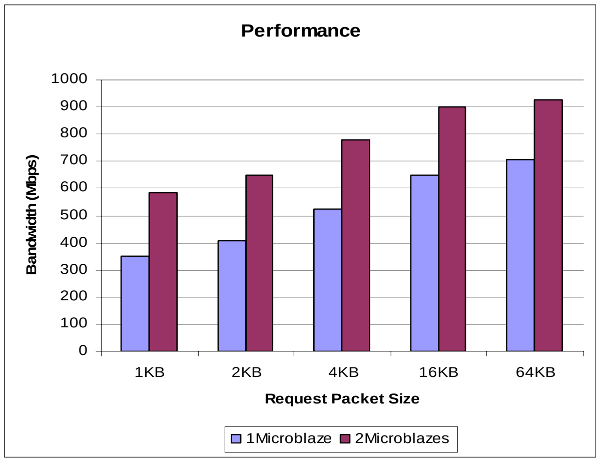


Figure 8. Performance of the systems

latency with the one based on the network processor but in our case the design can be extended with other payload processing modules such as encryption, compression or intrusion detection without wasting processor's cycles.

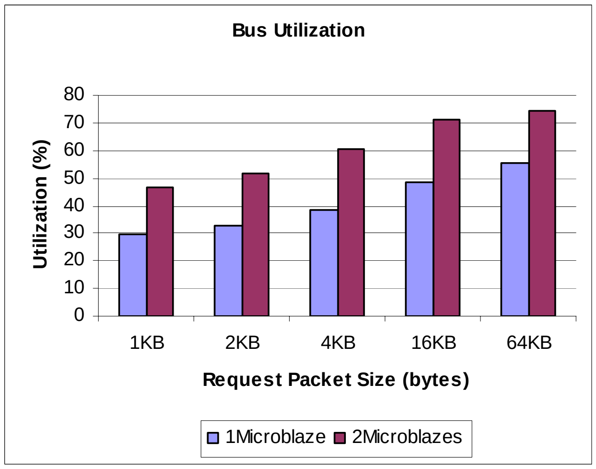


Figure 9. Bus Utilization

Table 4. Processing Latency for packets

Packet Type		Latency (us)		
		Reconf.	IXP2400[3]	Linux[3]
Control Packet	SYN	5.5	7.2	48
	ACK/Req.	8.8	8.8	52
	SYN/ACK	8.5	8.5	42
Data Packet	Data	6.9	6.5	13.6
	ACK	6.6	6.5	13.6

4.2 Area

Table 5 shows the area distribution of the system. As it is shown, the main constraint is the number of Block RAMs that have been used. This table shows also the number of equivalent gates. As it is shown the number of equivalent gates is much smaller than a typical network processor. As a figure of merit, the Agere's PayloadPlus Network Processor occupies 210 million transistors [13], thus about 35 million gates (assuming 1 gate = 6 transistors). Each MicroBlaze occupies 1088 Slices including the memory controllers and each Gigabit MAC occupies 790 slices. The main area constraint is the use of block RAMs which is 71% of the total block RAMs. The main advantage of the proposed scheme is that the remaining logic elements (slices) could be exploited for additional payload-processing modules (such as encryption, compression, or intrusion detection). Alternatively, the spare logic elements could be used to add SRAM or DRAM controllers that could increase the number of simultaneous connections or the size of the buffers. For example, the network processor-scheme ([3]) which uses the Intel IXP2400 is connected to a 8MB SRAM and a 128MB DRAM hence it can support more connections than our current scheme.

Table 5. Area allocation

Module	Number	Utilization
Slices	9847	32%
DSP48	6	9%
BRAM16	96	71%
Equivalent gates	301,948	

Figure 10 shows the area distribution by module. As it is shown in this figure, the CAM for the connections holds almost 31% of the total design area. This is also the main reason that the connection are stored in BRAMs using hash algorithms and only when there is a collision there are stored in the CAM. Each MicroBlaze occupies almost 11% of the area and each Gigabit Media Access Controller occupies 8%. The Connection Module and the URL Module occupies 9% and 10% respectively.

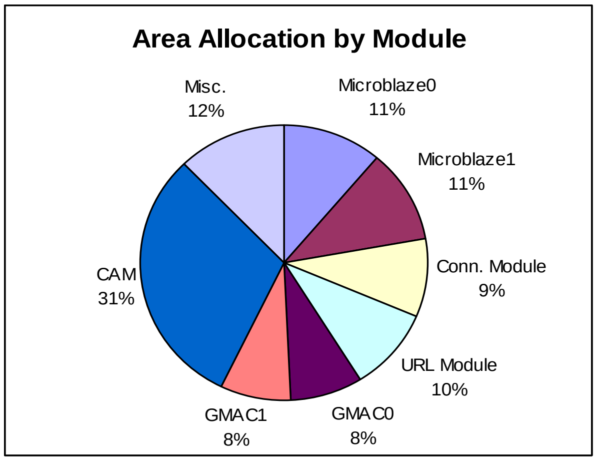


Figure 10. Area allocation by Module

4.3 Power

The main drawback of FPGAs is the power consumption. Hence, a detailed power analysis has been performed to identify the main sources of power consumption. The following design flow has been used. The system is synthesized, placed and routed using the Xilinx framework. Due to a lack of an evaluation board with 2 gigabit network ports the system has been evaluated from the placed and routed system. The design has been simulated using the Modelsim cycle-accurate simulator and the switching activity of the design has been extracted. The power consumption of the system has been estimated using the Xilinx's XPower tool. This tool compiles the design and the switching activity and reports the power consumption.

Figure 11 shows the power distribution by type of the system. According to this figure 42% of the power consumption is consumed by the signals. The clock consumes 29% of the overall power, while the logic consumes also 29% of the power.

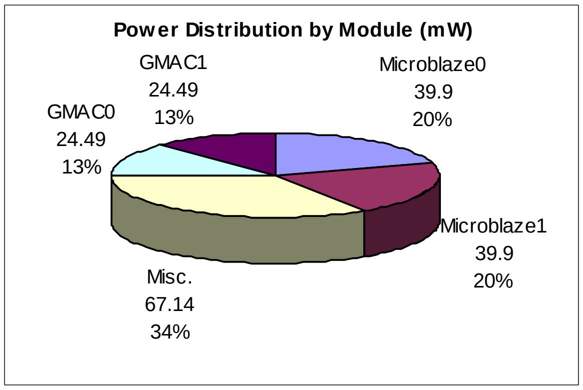


Figure 11. Power Distribution by Module

Figure 12 shows the dynamic power distribution by module. As it is shown, each processor consumes almost 20% the total power. Each Gigabit Media Access Controller consumes 13% of the power excluding the IO power, while the miscellaneous components (bus, co-processors) consume 34% of the power. Overall, the power consumption (including the core and the IO power consumption) is 870mW. The IO modules consume 235.4mW (operated at 2.5V) while the remaining power (634.6mW) is consumed by the internal modules (operated at 1.2V).

As a figure of merit, in [16] it is shown that the Intel IXP 2400 network processor consumes almost 13.3W for the IPv4 forwarding benchmark achieving 8Gbps forwarding throughput. The low power consumption in our scheme is mainly caused by the use of low frequency components. The typical network processor use micro-processors that operate in high frequency (e.g. the IXP 2400 micro-engines use 600MHz clock frequency). Since, the power consumption is proportional to the frequency the use of 100MHz clock frequency in our scheme keeps the power consumption to low level.

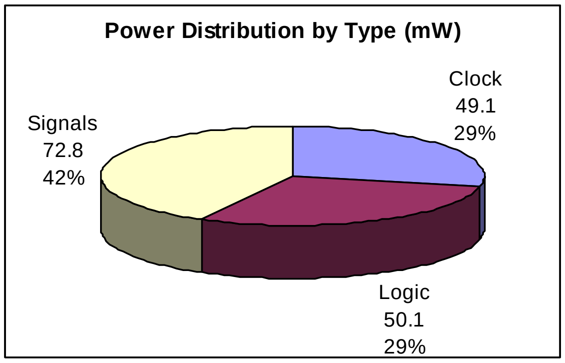


Figure 12. Power Distribution by type

5. CONCLUSIONS

The use of FPGAs is an appealing alternative to the use of specialized platform (such as network or digital signal processors) because of the flexibility they provide. On the other hand the main concern against the use of the FPGAs is the power consumption and the limited clock frequency. In this paper we have presented an efficient architecture for content-based switching that can be used to server cluster to provide higher response time to web users. The proposed scheme provides higher throughput compared to a mainstream network processor, while the power consumption is less than a network processor. This is due to the fact that specialized hardware coprocessors are used to accelerate the critical functions while the operating frequency of the system (which is the main source of power consumption) is held low (100 MHz). However, the flexibility is preserved using two soft-core 32-bit RISC processors customized for network processing instructions (barrel shifters, pattern comparison and checksum modules). Hence, we claim that a careful architecture in which the processor, the communication and the co-processors compose a balanced system, a viable and efficient system can be created that can perform better or equal to the current network processors for a limited number of connections. The proposed scheme can be extended using SRAM and DRAM controllers to support more connections.

Furthermore, the utilization of dynamically reconfigurable systems can further increase the performance and the flexibility of the proposed scheme. Dynamic reconfiguration provides the flexibility to add/remove more specialized co-processors during run-time based on the requirements of the application. Hence, in the future work we are investigating a content-based switch in which the co-processors will be loaded depending on the content switch configuration (URL-based, payload-based, cookie-based, etc.).

REFERENCES

[1] Z. Naseh, H. Khan, "Designing Content Switching Solutions", Cisco Press, ISBN: 1-58705-213-X

[2] L. Zhao, Y. Luo, L. Bhuyan, R. Iyer, "Design and Implementation of a Content-aware Switch using a Network Processor", Proceedings of the 13th Symposium on High Performance Interconnects (HOTI'05), August 2005, CA, USA

[3] L. Zhao, Y. Luo, L. Bhuyan, R. Iyer, "SpiceNP: A TCP Splicer using a Network Processor", Proceedings of the 1st Symposium on Architectures for Networking and Communications Systems (ANCS'05), October 2005, NJ, USA

[4] T. Spalink, S. Karlin, L. Peterson, Y. Gottlieb, "Building a Robust Software-Based Router Using Network Processors", Proceedings of the 18th ACM

symposium on Operating systems principles, pages 216 -229, 2001

[5] D. A. Maltz, P. Bhagwat, “TCP Splicing for Application Layer Proxy Performance”, IBM Research Report RC 21139, 1998

[6] A. Cohen, S. Rangarajan, H. Slye, “On the Performance of TCP Splicing for URL-aware Redirection”, In proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems, Boulder, CO, Oct. 1999

[7] “Analyzing the Performance of Web Switches”, Application Note, Foundry Networks

[8] A. Apostolopoulos, D. Aubespin, V. Peris, P. Pradhan, D. Saha, “Design, Implementation and Performance of a Content-Based Switch”, Proceedings of the Infocom 2000

[9] D. Maltz, P. Bhagwat, “TCP Splicing for Application Layer Proxy Performance”

[10] Linux Virtual Server Project, http://www.linuxvirtualserver.org

[11] A. Moestedt, P. Sjödin, “IP Address Lookup in Hardware for High-Speed Routing”, IEEE Hot Interconnects VI, pp. 31-39, Stanford, CA USA, August 1998

[12] Web caching, San Diego Super Computer Center Web Traces, http://www.web-caching.com

[13] “Agere Systems Network Processor Design”, Synopsis, Compiler Magazine, April 05

[14] “MicroBlaze Processor Reference Guide”, Xilinx Documentation, May 2005

[15] “Fast Simplex Link Bus v2.00”, Xilinx Documentation, April 2005

[16] “IXP2400 Intel Network Processor IPv4 Forwarding Benchmark Full Disclosure Report for Gigabit Ethernet”, Network Processing Forum, March 5, 2003

[17] “UCB-home-IP traces Nov 17”, UC Berkeley Home IP Web Traces , The Internet Traffic Archive

[18] B. C. Brodie, R. K. Cytron, D. E. Taylor, “A Scalable Architecture for High-Throughput Regular-Expression Pattern Matching”, The 33rd Annual International Symposium on Computer Architecture, June 2006, Boston, MA

[19] S. Fide, S. Jenks, “A Survey of String Matching Approaches in Hardware”, TR SPDS 06-01, University of California - Irvine, March 2006

[20] I. Sourdis, D.N. Pnevmatikatos, S. Wong, S. Vassiliadis, A Reconfigurable Perfect-Hashing Scheme for Packet Inspection**,** proceedings of 15th International Conference on Field Programmable Logic and Applications (FPL 2005), Tampere, Finland, August 2005