Performance Modeling and Analysis of Web Switches

Jie Lu
BMC Software Inc.
Waltham, MA 02451

Jie Wang
University of Massachusetts
Lowell, MA 01854

A Web switch is a key component in the Web cluster architecture. It is often built on a network processor (NP). This paper presents the first known analytic performance model for analyzing performance of NP-based Web switches. The model contains parameters of system configurations, dispatching algorithms, and workload characterizations, allowing users to troubleshoot performance bottlenecks and provision the capacity of the Web switch.

1. Introduction

1.1 Web clusters and Web switches

A Web cluster, also known as a Web farm, is a common architecture for providing Web services. It uses multiple Web servers to distribute workloads between individual servers (see Figure 1). Servers in a Web cluster are tightly coupled in a single location. The key component of the Web cluster architecture is a Web switch, acting as a dispatcher to route incoming requests to individual servers. Web clusters offer high scalability and availability. Unlike distributed Web systems, Web clusters are transparent to clients. Each server masks its IP address to its clients. Only the virtual IP address, which corresponds to the front-end Web switch, is visible to clients.

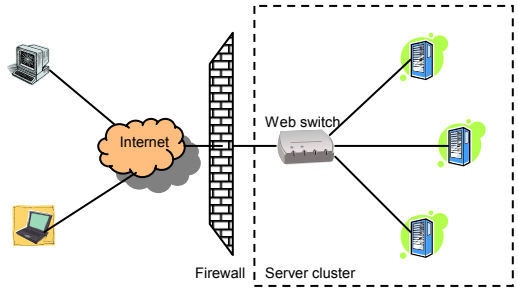

Figure 1: Web cluster architecture

A Web switch can be a layer-4 (L4) switch, a layer-7 (L7) switch, or a layer-4 and layer-7 combined switch (L4-7) that works at both layers. An L4 switch performs content-blind routing at the TCP layer. It determines the target server when a client requests a TCP connection with TCP SYN packet. It then assigns packets pertaining to the same connection to the same server. This mechanism is efficient, for packets do not go through the application level. However, it lacks the ability of dispatching requests according to contents.

An L7 switch, also known as a content-aware switch, operates at the application level. It provides application traffic management through deep packet inspections. An L7 switch performs the standard handshake procedure with clients at the client-side interface without any server involved. At the server-side interface, the switch keeps persistent TCP connection with each individual server, thus reducing connection establishment overheads. Figure 2 illustrates the difference between the traffic flows of an L4 switch and an L7 switch.

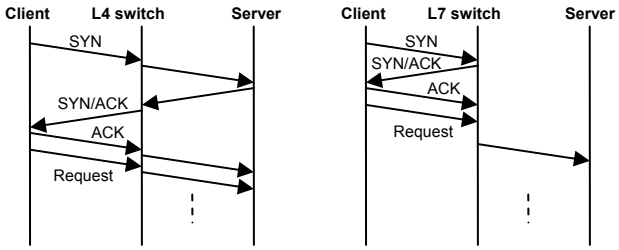

Figure 2: Comparison of L4 and L7 switches

A number of algorithms have been devised for L4 switches and L7 switches. They can be categorized as static algorithms, client state aware algorithms, server state aware algorithms, and client and server state aware algorithms. Descriptions of these algorithms can be found in a recent survey paper [1].

1.2 Network processors

Network processors (NP) are network devices specifically designed to store, process, and forward large volumes of data packets at wire speed with

strong programmability. They enable users to create and add, through software, the latest and best network services while maintaining high packet throughput and low packet latency. They offer both performance and flexibility through highly parallel, fully programmable architecture, differentiating them from general-purpose processors and hardware-based solutions, such as ASIC-based switches and routers. General-purpose processors offer programming flexibility, but they lack packet-processing performance. ASIC-based switches and routers offer packet-processing performance, but they have limited or no programmability.

A typical network processor consists of an array of programmable packet processors (PP) in a highly parallel architecture, a programmable control processor (a.k.a. a core processor), hardware coprocessors or accelerators for common networking operations, high-speed memory interfaces, and high-speed network interfaces. Examples of network processors include IBM's PowerNP [5], Intel's IXP [6], and Motorola's C-Port [7]. We will apply our performance model to analyze IXP2400-based Web switches. Figure 3 shows the functional components of IXP2400.

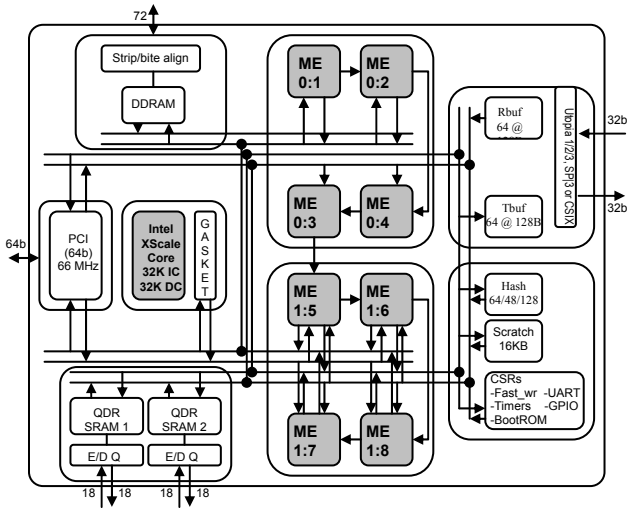

Figure 3: Intel IXP2400 block diagram, where ME stands for "micro engine" (i.e., packet processor)

2. Performance model

NP-based Web switch applications typically go through three major pipeline stages: packet receiving, packet processing, and packet transmitting (see Figure 4). Packet receiving and packet transmitting can each be implemented on a single packet processor or on multiple packet processors in parallel. Packet processing can be implemented on multiple packet processors in parallel or in pipeline.

We model the packet-flow process on a network processor as a queuing network (QN) and devise a performance model (see Figure 5). In this model we supply an input queue to each packet processor, memory unit, and special resource in this model. The control processor is not included in the model, for it is normally used to perform configuration management and exception handling, rather than packet processing. In particular, this model assumes that packet receiving and packet transmitting are each handled by a packet processor, and packet processing is handled by multiple packet processors in parallel. While there are other ways to allocate resources, we want to keep our model as simple as possible. We first measure service rate, throughput, and response time at the component level, using the Mean Value Analysis (MVA) method. We then measure throughput and response time at the system level by combining measuring results at the component level.

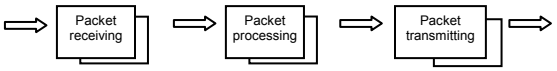

Figure 4: Packet flow in network processor

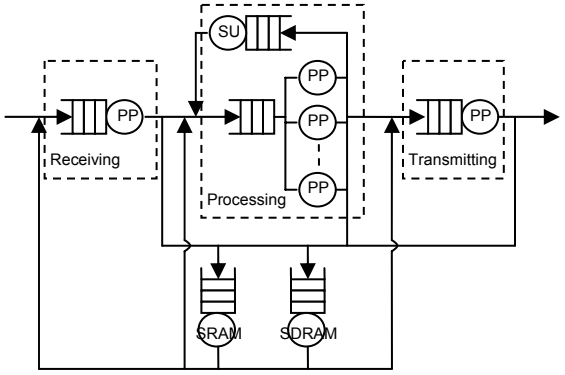

Figure 5: Performance model of network processor

It is relatively easy to measure the performance of the packet receiving stage, for the packet receiving component consists of only one packet processor and two memory units. But there is a subtlety: A packet processor contains multiple hardware threads, and so it cannot be treated as a load-independent resource. We model multiple threads in a packet processor using only load-independent resources in a closed QN model (see Figure 6). In this model, the number of threads becomes the number of requests. We then use the MVA algorithm [3] to measure throughput as follows:

Residence time at resource (queue) *i* with *n* requests in QN:

$$r_i(n) = D_i \times \left[1 + f_i(n-1)\right], \qquad (2\text{-}1)$$

where D_i is the service demand at resource *i* and $f_i(n)$ is the average number of requests in queue *i* with *n* requests in QN.

Throughput of QN with *n* requests in QN:

$$X(n) = \frac{n}{\sum_{i=1}^{K} r_i(n)}, \tag{2-2}$$

where *K* is the number of resources.

Queue length at resource *i* with *n* requests in QN:

$$f_i(n) = X(n) \times r_i(n). \tag{2-3}$$

Thus, we can recursively apply these formulas to calculate the throughput $X(n)$ of the closed model of QN. This allows us to model each multi-threaded processor as a single threaded processor with variable service rate.

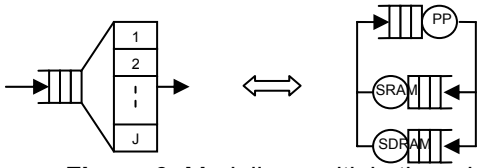

Figure 6: Modeling multiple threads in packet processor

Now we consider the packet receiving stage as an open QN model (Figure 5) with unbounded queues and variable service rate. The service rate is given by $X(n)$ we have derived above. In this model, the average number of requests is given by the following formula given in [3]:

$$\overline{N} = \left[1 + \sum_{k=1}^{K} \frac{\lambda^k}{\beta(k)} + \frac{\lambda^K}{\beta(K)} \frac{\rho}{1-\rho}\right]\left[\sum_{k=1}^{K} \frac{k\lambda^k}{\beta(k)} + \frac{\rho\lambda^K[\rho + (J+1)(1-\rho)]}{(1-\rho)^2 \beta(K)}\right], \tag{2-4}$$

where λ is the request arrival rate, $\beta(k) = X(1) \times X(2) \times \cdots \times X(k)$, and $\rho = \lambda / X(K)$.

Finally, the response time of the packet receiving stage can be obtained from Little's Law:

$$R_{PR} = \frac{\overline{N}}{X_{PR}}. \tag{2-5}$$

where X_{PR} is the throughput of the packet receiving stage. According to the flow equilibrium principle, the throughput equals to the arrival rate, i.e., $X_{PR} = \lambda$. However, the maximum throughput is bounded above by $X(K)$.

We model the performance of the packet transmitting stage in the same way as we model the performance of the packet receiving stage.

In the packet processing stage, multiple packet processors are used to handle requests (service demands) from a single queue. A queue with *m* resources and service demand *D* at each resource can be replaced in the queuing network with two resources in tandem (see Figure 7). One is a load-independent or a load-dependent resource with D/m service demand, and the other is a delay resource (without queue) with $D \times (m-1)/m$ service demand. This allows us to model multiple processors as a single processor with a delay resource.

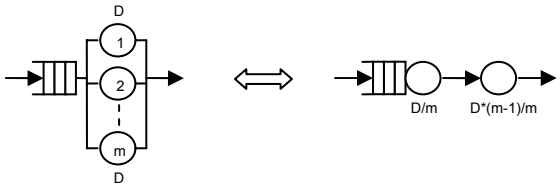

Figure 7: Model parallel packet processors

Similar to the packet receiving stage, we measure the service rate, the throughput, and the response time in the packet processing stage using the MVA algorithm on the closed QN model (see Figure 8).

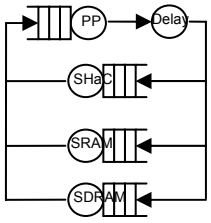

Figure 8: Closed model for the packet processing stage

Finally, we can measure the throughput and the response time of the whole network processor as

$$X_0 = \min(X_{PR}, X_{PP}, X_{PT}), \tag{2-6}$$

$$R = R_{PR} + R_{PP} + R_{PT}. \tag{2-7}$$

3. Performance analysis of Web switches

We now measure performance of NP-based Web switches using the performance model we have devised. For an NP-based Web switch, it is easy to obtain the request arrival rate, λ, and the average packet size, *m*. But it is difficult to measure the average service demand for each request at each resource. This is because measuring the utilization of each packet processor is difficult. In this paper, we estimate service demand by analyzing the pseudo code of each application. Packet processors in IXP2400 are RISC processors, and so most instructions only take one clock cycle. Thus, service demand for a packet processor can be obtained based on the number of instructions for processing each request. For SRAM and SDRAM, the service demand can be obtained based on the number of memory reference made for each request.

$$D_{SRAM} = \frac{reference\_count \times SRAM\_latency}{clock\_rate}, \quad (3\text{-}1)$$

$$D_{SDRAM} = \frac{reference\_count \times SDRAM\_latency}{clock\_rate}, \quad (3\text{-}2)$$

$$D_{SHaC} = \frac{reference\_count \times hash\_latency}{clock\_rate}, \quad (3\text{-}3)$$

$$D_{PR} = \frac{instruction\_count}{clock\_rate}, \quad (3\text{-}4)$$

$$D_{PT} = \frac{instruction\_count}{clock\_rate}, \quad (3\text{-}5)$$

$$D_{PP} = \frac{instruction\_count}{clock\_rate \times processor\_count}, \quad (3\text{-}6)$$

$$D_{Delay} = \frac{instruction\_count \times (processor\_count - 1)}{clock\_rate \times processor\_count}, \quad (3\text{-}7)$$

Pseudo code, clock rates, and memory latency are NP-dependent. As an example, we apply our performance model to analyze performance of a Web switch built on an Intel IXP2400 network processor. Performance of a Web switch built on an NP with similar architecture as IXP can be analyzed using the same method. Listed below are a number of critical hardware parameters of IXP2400:

- Microengine (i.e. packet processor) clock rate: 600 MHz
- The number of hardware threads in a microengine: 8
- SRAM latency: 90 clock cycles
- SDRAM latency: 120 clock cycles
- SHaC (Scratch, Hash) latency: 16 clock cycles

3.1 Packet receiving and packet transmitting

We first investigate the packet receiving stage and the packet transmitting stage, for they are the same regardless what Web switch dispatching algorithms are used. For these two stages, we follow the standard code given in [2]. We assume that the average packet size is 1024 bytes, decomposing to 16 mpackets by IXP [2]. Each mpacket contains 64 bytes. IXP receives a packet by reading one mpacket at a time (instead of one byte) and assembles mpackets back to the packet.

According to the standard code given in [2], when a packet arrives it takes 864 cycles of instructions, 1 scratch pad (SHac) I/O, 1 SRAM I/O, and 16 SDRAM I/O to receive and assemble the packet at the receiving stage. Table 1 gives the breakdown for instructions count and memory references for packet receiving stages. All initializations occur only at start time are not included here.

Steps	Inst. cycles	Scratch Pad	SRAM	SDRAM	Use per pkt.
Buffer allocation	14		1		1
Prep. trd for an mpacket to arrive	18				16
Inter-trd signals per mpacket	8				16
Move the mpacket into DRAM	26			1	16
Inter-trd signals per packet	4				1
Enqueue the packet for processing	14	1			1

Table 1: Microcode analysis for packet receiving stage

It is straightforward to see that each HTTP request involves 6 messages between the client and the server. For simplicity, we use a single packet processor to handle both incoming and outgoing packets. (Implementations that use one packet processor for incoming traffic and a separate packet processor for outgoing traffic can be modeled using the same method we present here.) Thus, we can calculate the service demand for each request at the packet receiving processor as follows:

$$D_{PR} = \frac{864}{600{,}000{,}000} \times 6 = 0.0086ms$$

$$D_{SHaC} = \frac{1 \times 16}{600{,}000{,}000} = 0.00003ms$$

$$D_{SRAM} = \frac{1 \times 90}{600{,}000{,}000} \times 6 = 0.0009ms$$

$$D_{SDRAM} = \frac{16 \times 120}{600{,}000{,}000} \times 6 = 0.0192ms$$

Applying these service demands to the model described in Figure 6, we obtain the throughput of the packet receiving stage in Table 2.

n	r_{pr}	r_{sram}	r_{sdram}	r_{shac}	X	n_{pr}	n_{sram}	n_{sdram}	n_{shac}
0	0.00	0.00	0.000	0.00	0.00	0.00	0.00	0.00	0.00
1	8.60	0.90	19.20	0.03	34806.82	0.29	0.03	0.66	0.00
2	11.17	0.92	32.03	0.03	45286.03	0.50	0.04	1.45	0.00
3	12.95	0.93	47.05	0.03	49203.98	0.63	0.04	2.31	0.00
4	14.08	0.94	63.65	0.03	50824.65	0.71	0.04	3.23	0.00
5	14.75	0.94	81.31	0.03	51525.58	0.76	0.04	4.19	0.00
6	15.13	0.94	99.64	0.03	51834.69	0.78	0.04	5.16	0.00
7	15.34	0.94	118.36	0.03	51972.20	0.79	0.04	6.15	0.00
8	15.46	0.94	137.31	0.03	52033.60	0.80	0.04	7.14	0.00

Table 2: Throughput of QN for packet receiving stage

The relationship between the arrival rate and the response time at this stage is calculated using the average number of requests and response time described in Section 3 (see Figure 9), where the arrival rate is measured by the number of packets per second.

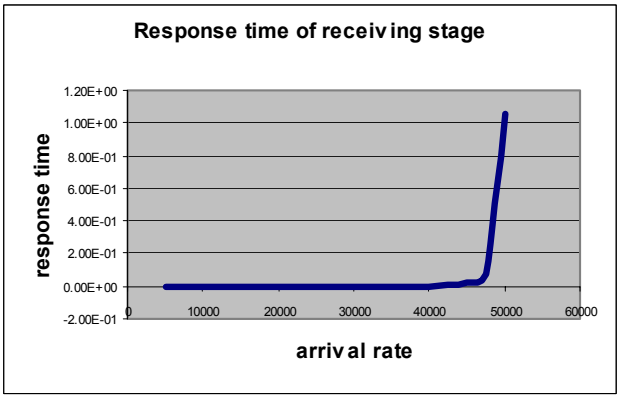

Figure 9: Response time at the packet receiving stage

According to [2], the packet transmitting stage takes 1541 cycles of instructions, 1 Scratch pad I/O, 1 SRAM I/O, and 16 SDRAM I/O to transmit one packet. Table 3 gives the breakdown for instructions count and memory references for packet receiving stages. Thus, we can calculate the service demand for each request at the packet transmitting processor as follows.

$$D_{PT} = \frac{1541}{600{,}000{,}000} \times 6 = 0.0154ms$$

$$D_{SHaC} = \frac{1 \times 16}{600{,}000{,}000} = 0.00003ms$$

$$D_{SRAM} = \frac{1 \times 90}{600{,}000{,}000} \times 6 = 0.0009ms$$

$$D_{SDRAM} = \frac{16 \times 120}{600{,}000{,}000} \times 6 = 0.0192ms$$

Steps	Inst. cycles	Scratch Pad	SRAM	SDRAM	Use per pkt
Inter-trd signals	5				16
Update TBUF in flight	12				16
Update segmentation state	56	1			16
Transfer mpacket into TBUF	13			1	16
Validate TBUF	8				16
Update local transmit state	2				16
Free buffer	5		1		1

Table 3: Microcode analysis for packet transmitting stage

Similar to the packet receiving stage, we obtain the throughput of the packet transmitting stage in Table 4.

n	r_{pt}	r_{sram}	r_{sdram}	r_{shac}	X	n_{pr}	n_{sram}	n_{sdram}	n_{shac}
0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
1	15.40	0.90	19.20	0.03	28145.22	0.43	0.02	0.54	0.00
2	22.07	0.92	29.57	0.03	38020.49	0.83	0.03	1.12	0.00
3	28.32	0.93	40.79	0.03	42810.22	1.21	0.04	1.74	0.00
4	34.07	0.93	52.72	0.03	45574.89	1.55	0.04	2.40	0.00
5	39.31	0.93	65.33	0.03	47338.58	1.86	0.04	3.09	0.00
6	44.06	0.94	78.58	0.03	48536.79	2.13	0.04	3.81	0.00
7	48.33	0.94	92.43	0.03	49386.03	2.38	0.04	4.56	0.00
8	52.16	0.94	106.84	0.03	50006.15	2.60	0.04	5.34	0.00

Table 4: Throughput of QN for packet transmitting stage

The relationship between the arrival rate and the response time at the packet transmitting stage is shown in Figure 10, where the arrival rate measures the number of packets arrived per second.

3.2 Packet processing

Assume that the remaining packet processors can be used to implement Web switch dispatching algorithms. We choose the Least Loaded (LL) policy for L4 switches and Locality-Aware Request Distribution

(LARD) [4] policy for L7 switches. Other schemes can be modeled easily with minor modifications.

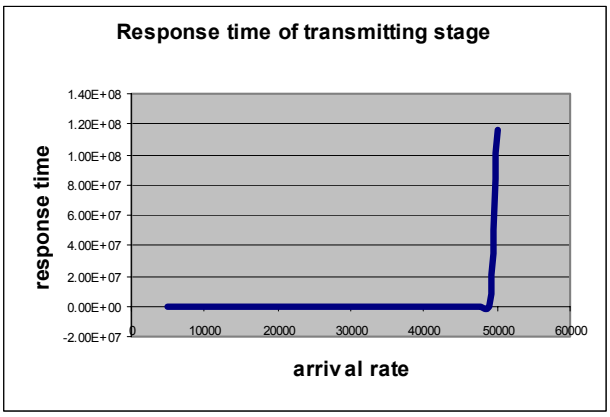

Figure 10: Response time at the packet transmitting stage

3.2.1 L4 switch

Listed below are the primary steps for processing an incoming packet at an L4 switch.

	Steps	Inst. cycles	SHaC	SRAM	SDRAM
1	Dequeue a packet from source	46	1		1
2	Validate Ethernet packet	53		1	1
3	Strip Ethernet header	19			1
4	Retrieve IP header	104			1
5	Retrieve TCP header	104			1
6	Generate hash value	12	1		
7	SYN: Get least load server	60		2	
8	SYN: Add binding entry	100		4	
9	FIN and REQ: Search binding table	31		2	
10	REQ: Get destination server	50		1	
11	FIN: Remove binding entry	80		2	
12	Update IP header	26			1
13	Add Ethernet header	123		2	2
14	Enque the packet for transmit	12	1		

Table 5: Pseudo code analysis for L4 packet processing

Processing an outgoing packet is much simpler, for the only processing involved is masking the source address. The above steps can be reused by eliminating step 5 through step 11.

Note that the error condition handlings are not listed here.

We compose IXP2400 pseudo code for these steps and estimate service demand at each step as shown in Table 5.

According to our estimates, the service demand for incoming and outgoing packets consists of 2998 cycles of instructions, 15 SHaC access, 31 SRAM I/O, and 45 SDRAM I/O. Thus, we can calculate the service demand for each request at L4 switch processing stage as following:

$$D_{PP} = \frac{2998}{600{,}000{,}000} = 0.0050ms$$

$$D_{SRAM} = \frac{31 \times 90}{600{,}000{,}000} = 0.0047ms$$

$$D_{SDRAM} = \frac{45 \times 120}{600{,}000{,}000} = 0.0090ms$$

$$D_{SHaC} = \frac{15 \times 16}{600{,}000{,}000} = 0.0004ms$$

Suppose that 4 packet processors are used for dispatching packets in parallel. Applying the above parameters to the model in Figure 8, we obtain the throughput at the packet dispatching stage in Table 6. The relationship between the response time and the arrival rate is shown in Figure 11.

n	r_{PP}	r_{SRAM}	r_{SDRAM}	r_{SHaC}	X	n_{PP}	n_{SRAM}	n_{SDRAM}	n_{SHaC}
0	0.00	0.00	0.00	0.00	0	0.00	0.00	0.00	0.00
1	1.25	4.70	9.00	0.40	65147	0.08	0.31	0.59	0.03
2	1.35	6.14	14.28	0.41	90179	0.12	0.55	1.29	0.04
3	1.40	7.30	20.59	0.41	100988	0.14	0.74	2.08	0.04
4	1.43	8.17	27.71	0.42	106041	0.15	0.87	2.94	0.04
5	1.44	8.77	35.45	0.42	108522	0.16	0.95	3.85	0.05
6	1.45	9.17	43.62	0.42	109775	0.16	1.01	4.79	0.05
7	1.45	9.43	52.10	0.42	110417	0.16	1.04	5.75	0.05
8	1.45	9.60	60.77	0.42	110750	0.16	1.06	6.73	0.05

Table 6: Throughput of QN for L4 processing

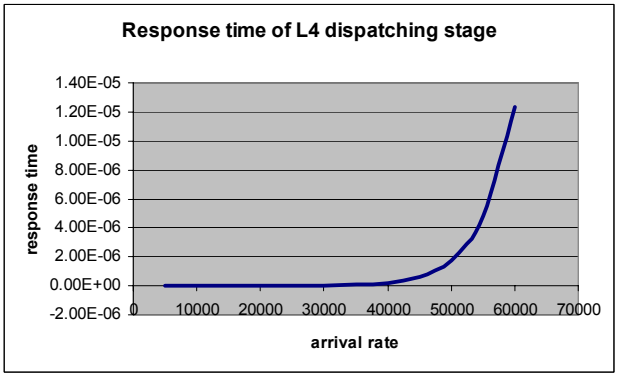

Figure 11: Response time at the dispatching stage on an L4 switch

3.2.1 L7 switch

Packet processing on an L7 switch differs from an L4 switch in a number of ways. An L7 switch acts as a proxy between clients and the Web cluster. It makes handshake with clients without any server involved. Table 7 lists the primary steps for processing an incoming packet.

	Steps	Inst. cycles	SHaC	SRAM	SDRAM
1	Dequeue a packet from source	46	1		1
2	Validate Ethernet packet	53		1	1
3	Strip Ethernet header	19			1
4	Retrieve IP header	104			1
5	Retrieve TCP header	104			1
SYN6	Create handshake packet	24		1	1
SYN7	Add Ethernet header	123		2	2
SYN8	Enque the packet for transmit	12	1		
SYN9	Generate hash of client info	12	1		
SYN10	Add client entry	120		4	
ACK6	Generate hash of client info	12	1		
ACK7	Get client entry	70		3	
ACK8	Update client status	30		1	
REQ6	Parse client request	960		10	16
REQ7	Generate hash of target object	12	1		
REQ8	Get target server from binding table	80		3	
REQ9	Update IP header	26			1
REQ10	Add Ethernet header	123		2	2
REQ11	Enque the packet for transmit	12	1		
FIN6	Generate hash value of client info	12	1		
FIN7	Remove client entry	80		2	

Table 7: Pseudo code analysis for L7 incoming packet processing

	Steps	Inst. cycles	SHaC	SRAM	SDRAM
1	Dequeue a packet from source	46	1		1
2	Validate Ethernet packet	53		1	1
3	Strip Ethernet header	19			1
4	Retrieve IP header	104			1
5	Update IP header	26			1
6	Add Ethernet header	123		2	2
7	Enque the packet for transmit	12	1		

Table 8: Pseudo code analysis for L7outgoing packet processing

An L7 switch keeps persistent TCP connection with each individual server in the cluster, reducing handshake overheads. Processing response packets from a server is straightforward as listed in Table 8.

We again compose IXP2400 pseudo code for these steps and estimate server demand at each step as shown in Table 7 and Table 8. According to our estimates, the service demand for each request is 3835 cycles of instructions, 40 SRAM I/O, 58 SDRAM I/O, and 15 SHaC accesses. Thus,

$$D_{PP} = \frac{3835}{600{,}000{,}000} = 0.0064ms$$

$$D_{SRAM} = \frac{40 \times 90}{600{,}000{,}000} = 0.0060ms$$

$$D_{SDRAM} = \frac{58 \times 120}{600{,}000{,}000} = 0.0116ms$$

$$D_{SHaC} = \frac{15 \times 16}{600{,}000{,}000} = 0.0004ms$$

Applying the above parameters to the model in Figure 8, we obtain L7 throughput in Table 9. The relationship between the response time and the arrival rate is shown in Figure 12.

n	r_{PP}	r_{SRAM}	r_{SDRAM}	r_{SHaC}	X	n_{PP}	n_{SRAM}	n_{SDRAM}	n_{SHaC}
0	0.00	0.00	0.00	0.00	0	0.00	0.00	0.00	0.00
1	1.60	6.00	11.60	0.40	51020	0.08	0.31	0.59	0.02
2	1.73	7.84	18.47	0.41	70321	0.12	0.55	1.30	0.03
3	1.79	9.31	26.66	0.41	78585	0.14	0.73	2.10	0.03
4	1.83	10.39	35.91	0.41	82420	0.15	0.86	2.96	0.03
5	1.84	11.14	45.93	0.41	84289	0.16	0.94	3.87	0.03
6	1.85	11.63	56.51	0.41	85226	0.16	0.99	4.82	0.04
7	1.85	11.95	67.46	0.41	85702	0.16	1.02	5.78	0.04
8	1.85	12.14	78.67	0.41	85947	0.16	1.04	6.76	0.04

Table 9: Throughput of QN for L4 processing

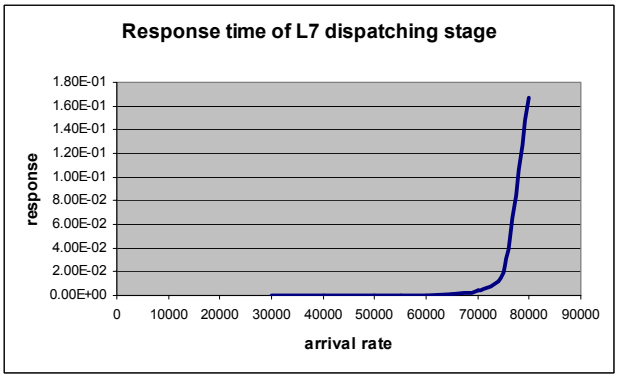

Figure 12: Response time at the dispatching stage on an L7 switch

4. Conclusion

We present a performance model for network processors and a theoretical method for analyzing NP-based Web switch performance. Results of such analysis may be used to help configure Web clusters. They may also be used for Web switch developers to better allocate resources.

References

[1] V. Cardellini, E. Casalicchio, M. Colajanni, and P. S. Yu. The state of the art in locally distributed web-server systems. ACM Computing Surveys, 34(2):263-311, 2002.

[2] E. Johnson and A. Kunze. IXP2400/2800 Programming: The Complete Microengine Coding Guide. Intel Press, 2003.

[3] D. A. Menasce and V. A. F. Almeida. Capacity Planning for Web Services: metrics, models, and methods. Prentice Hall, 2002.

[4] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum. Locality-aware request distribution in cluster-based network servers. In Proc. of the 8th ACM Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, Oct. 1998.

[5] IBM PowerNP Network Processors. http://www-3.ibm.com/chips/techlib/techlib.nsf/productfamilies/PowerNP_Network_Processors.

[6] Intel Network Processors, http://www.intel.com/design/network/products/npfamily/

[7] Motorola C-Port Network Processors, http://e-www.motorola.com/webapp/sps/site/homepage.jsp?nodeId=03DnXMx1Ks