

# Solving the Edge Server Streaming Bottleneck with the Separation Principle

Jordi Ros, Calvin Shen, Kevin Phan, Koji Tsuboi, Rod Allen,  
Henry Luk  
TCP/IP Acceleration Group, Xiran Labs  
17770 Cartwright, Irvine, California  
jros@xiran.com

Wei K. Tsai  
Department of Electrical and Computer Engineering  
University of California, Irvine  
wtsai@uci.edu

**Abstract** – The well-known edge server bottleneck is shown in this paper to be the result of a mismatch between the general-purpose architecture and the special-purpose data functions it was not intended to perform exclusively. Six overheads are identified that contribute significantly to the bottleneck. To solve this bottleneck problem, a solution is to apply the principle of separation between control and data functions. While the idea is not new, the application of this principle to the edge server architecture is novel in the convergence of three technologies: network, server, and storage. Notable performance has been obtained with this approach using ASIC implementation for TCP or UDP streaming applications.

## I INTRODUCTION

The advent of the Internet has brought us the convergence of three major technologies: networking, computing, and storage. At the heart of this convergence, sits the edge server, which is currently suffering from a severe bottleneck. This bottleneck, as in most cases, is the result of a speed mismatch between components. Instead of looking at the actual speeds, it is instructive to consider technology improvement velocities, as given in the table below.

Table 1 Technology trends

Technologies	Improvement velocities
Processing Capacity (Moore's Law)	2x/18 months
Data Storage Capacity	2x/9 months
Line Capacity (DWDM)	2x/7 months

Obviously, the processor is in a constant danger of being the bottleneck. This mismatch is routinely resolved by parallelization and other acceleration techniques, since the early days of computing. However, the edge server throughput still developed into a stubborn bottleneck that refuses to respond to acceleration techniques. For example, while Gbps network interface is now standard equipment, the best server today can hardly deliver video streaming traffic at a mundane speed of 150 Mbps.

To understand this bottleneck, one has to look inside the server, beyond the network and storage interface. High-end servers today are built around a classic architecture developed 50 years ago. Being built for general-purpose computation, the server is not optimized for Web specific functions, resulting in poor data throughput. In fact, six overheads can be identified as the culprit of the server bottleneck; most of them are the result of

the mismatch between the intended general-purpose and specialized data related functions.

The current solutions to the server bottleneck problem are three: *farming*, *clocking*, and *specialization*. In *farming*, the approach is to parallelize in space by adding hardware. This approach has produced multi-tiered groups of servers at data centers. This solution is obviously not scalable as it dramatically increases the cost associated with power, labor, software, and management. The law of diminishing return implies the farming approach is not price-performance scalable.

In *clocking*, the approach is to accelerate in time by boosting the clock speed. This approach is fundamentally the same as *farming*, except parallelization is done in the time domain. However, due to the six overheads described below, the server performance still scales poorly with the clock speed.

The only sensible solution is *specialization* where dominant specialized functions are optimized by specialized hardware. A logical consequence of specialization is *separation of distinct functions*. In fact, we have scores of successes in the tech world due to the *separation principle*. In networking, this principle first appeared in the form of control plane and data plane separation in the voice network. In computing, this principle first appeared in data and instructions separation in the Turing Machine. Most recently, this principle appears in the IETF working group - ForCES (Forwarding and Control Element Separation).

This paper will show that the separation principle has yielded a new architecture for the edge server. Unmatched performance has been obtained in both the laboratory and commercial settings.

## II VIDEO STREAMING EDGE SERVERS: CURRENT SCALABILITY ISSUES

While Internet servers have been improved throughout the last decade to accommodate the new performance requirements at both storage and network sides, today their design is still based on the classic generic purpose architecture (GPA) shown in Figure 1 [1]. The generality of the GPA comes at a cost in performance degradation due to six overheads: extra data copies, MTU inefficiencies, excessive I/O interrupts, excessive context switching, expensive software operations, and parallelization penalty. This section presents a brief explanation of these overheads.

### A. Extra data copies

Operating systems organize main memory into two areas: the user and kernel spaces. User applications can only write in the user space. If a service from the kernel space is needed, the request must be copied from the user space to the kernel space. This technique prevents faulty applications from overwriting the kernel code, which provides a level of protection. Two examples of kernel services that are protected with this mechanism are file system and networking services, both operated through the socket layer [2]. Considering the case of sending a piece of data  $D$  from storage to network, the following transactions are to occur:

1. *Invoke the file system services.* Data  $D$  is retrieved from the storage device and moved to the kernel space.
2. *Copy data.* Data  $D$  in the kernel space is copied to the user space.
3. *Copy data.* Data  $D$  in the user space, after being processed by the application, is copied back to the kernel space.
4. *Invoke network services.* Data  $D$  is sent from the kernel space to the network via the network device.

Steps 2 and 3 are an overhead incurred to satisfy the generic purpose of the GPA that requires protection against faulty applications. This overhead is especially significant in video streaming applications where large amounts of data are moved from storage to network.

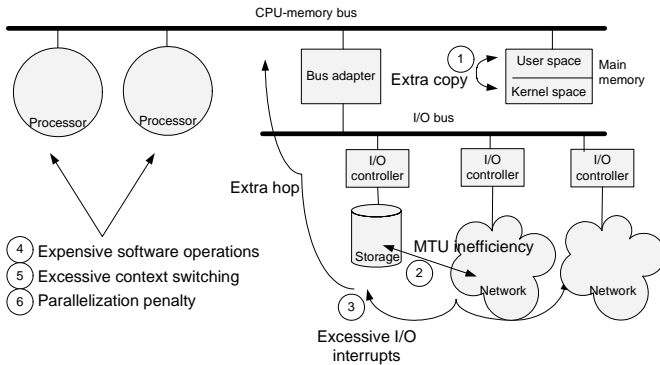


Figure 1 Generic purpose architecture and its six overheads

### B. MTU constraints

Packets transmitted on the wire are limited by constraints imposed at the physical layer. One such constraint is given by the maximum transmission unit (MTU). The MTU affects the overall performance of the system in two manners:

- 1- *Packet size upper bound.* Packets cannot be built that exceed the MTU. Since a major part of the processing cost is incurred on a per packet basis rather than on a per byte basis, the lower the MTU the bigger the processing overhead is.
- 2- *Kernel page size mismatch.* Information is retrieved and stored to storage devices in chunks of continuously located data, the so-called file system pages. Assume a typical file system page size of 4 Kbytes and a typical MTU of 1.5 Kbytes. In order to transmit a page, the operating system must build three different network descriptors (also known as DMA

descriptors [2]) of 1.5 Kbytes, 1.5 Kbytes and 1 Kbytes, respectively. Instead, if the MTU was unbounded, the page could be transmitted using a single descriptor. The mismatch between the page size and the MTU forces the system to allocate additional network descriptors.

While the MTU is a constraint imposed by the physical layer, *TCP segmentation* is a technique that virtually removes its downside effects. With TCP segmentation, packets are created at the TCP layer with a total size that is a multiple of the kernel page size. Then, at the physical layer an ASIC engine fragments this packet at wire speed into smaller packets of MTU bytes and regenerates a new TCP header for each of them. This technique allows transport layers to see virtually unbounded MTUs.

### C. I/O bus interrupts

The GPA scales in the number of supportable devices via a shared I/O bus. Examples of devices that can be attached to this bus are network devices, storage devices, sound cards and graphic cards. The flexibility of this design comes at a performance cost:

1- *Excessive contentions in the bus.* Servers attach both storage and network devices to the bus. Such devices generate asynchronous traffic that contends for bus resources. The result of these contentions is degradation of the overall performance as the storage and network traffic increases.

2- *Extra hop delay to reach the processor.* Packets from the network have to travel an extra hop (the I/O bus) in order to reach the processor resources.

### D. Context switching

Some applications require the execution of a set of services concurrently. Because the number of services is in general larger than the number of processors in the system, the operating system provides a task scheduler [2] that defines when each task is to be executed by what processor. In what follows, assume without loss of generality a GPA with a single processor. Let  $T_i$  be a task and let  $W_i$  be the set of operations to be carried out by this task. Suppose  $T_i$  is currently being executed by the processor and assume that the scheduler decides to preempt at time  $t$  this task so that another task  $T_j$  can be carried out. In GPA, the scheduler has no information regarding the nature of task  $T_i$  and, therefore, one should expect in general that at time  $t$  work  $W_i$  is still not finished. If that is the case, then the current work status for task  $T_i$  must be remembered so that the processor can continue to work on it at some future time. The operation of saving such information is known as *context switching*.

The context switching operation is an overhead due to the GPA. One can see that by designing a scheduler that is aware of the nature of task  $T_i$ , the context switching overhead can be removed. We call such type of scheduler a *run-to-completion (R2C) scheduler*. In a R2C scheduler, tasks are not switched until they are completed. By doing so, no task context needs to be remembered when a new task is loaded to the processor and, therefore, context switching overheads are eliminated.

### E. Software inefficient computations

Due to the popularity of the GPA, servers today perform many computationally intensive operations. These operations typically incur a large overhead. We name two examples:

1- *Checksum operations*. When computed by the processor, packet checksums become an expensive operation because they need to operate on all the bytes of the payload.

2- *Encryption algorithms*. As shown in [3], overheads due to software encryption can slow the servers by two orders of magnitude.

One common approach used to accelerate computational intensive operations such as the above mentioned is to attach specialized hardware engines to the I/O bus. Examples are network interface cards with checksum capability or encryption cards. Such approaches though, have two weaknesses:

1. They continue to suffer from the I/O bus overhead (section C. ).
2. The higher the layer of these operations, the more overheads one must incur in order to interface them. For instance, Secure Socket Layer (SSL) is a protocol that performs data encryption on top of TCP. In order to perform SSL, the GPA processors must first decode the TCP headers sitting at the kernel space. Because the hardware engines capable of decrypting the data are attached to the I/O bus, additional communication overheads are incurred due to the extra hop problem (section C. ).

### F. Parallelization penalty

Physical properties at the transistor level dictate that it is technically easier to build in the same silicon two processors running at a clock rate of  $R$  Hz than it is to build a single processor running at a clock rate of  $2R$  Hz. Based on this principle, computer architects find arbitrage opportunities to scale a system to higher performance levels [4]. Nevertheless, because the GPA hides the nature of the running applications from the processors, additional overheads need to be incurred in the parallel architecture. This is mainly because the processors need to lock memory locations, which leads to memory contentions. Therefore, the effect of adding  $n$  processors to perform a particular task using a GPA leads to a maximum performance improvement factor of  $n \cdot r_{gpa}$ , with  $0 < r_{gpa} < 1$ . For instance, measurements in [7] indicate that typically  $r_{gpa} < 0.5$ .

By designing a parallel processor architecture that is aware of the type of tasks, one can remove the above-mentioned overheads and achieve improvement factors of  $n \cdot r_{aware}$ , with  $r_{gpa} < r_{aware} < 1$  and  $r_{aware}$  close to unity.

## III NEW EDGE SERVER ARCHITECTURE FOR STREAMING TRAFFIC

This section introduces a new edge server architecture for content delivery networks. This architecture aims to resolve the six server overheads in the GPA. While each overhead is overcome via some specific mechanism (as explained later in section 0), the fundamental modification that the new

architecture brings is a clean separation of control plane and data plane.

### A. Architecture: separation of control plane and data plane

Figure 2 presents a model that explains the current edge server. This model emphasizes two aspects. First, the clear separation among modules that lay below the I/O bus (network and storage cards) and above the I/O bus (the main memory and processors). Second, while physically each network and storage port admits data in both incoming (towards the server) and outgoing (from the server) directions, the model presents a logical separation of the input and output functions, adopting the convention that data flows from left to right. Figure 2 also plots the paths traced by control and data planes, showing that current servers make no distinction between them. For instance, let us consider the case of streaming video via RTCP/RTP. RTCP is responsible for providing control over the streamed data (e.g. QoS management) while RTP delivers the actual content (e.g. a movie with audio and video bands). In the current architecture, RTCP and RTP packets traverse the same modules. On the incoming path, packets arrive from a network interface, cross the I/O bus, and are stored in main memory where they are interpreted by the processors. The symmetric case occurs for data that are retrieved from the storage device (or from another network device if the server acts as a cache proxy). Data arrive from the storage device, crosses the I/O bus, and is temporarily stored in the main memory waiting to be sent out to the network.

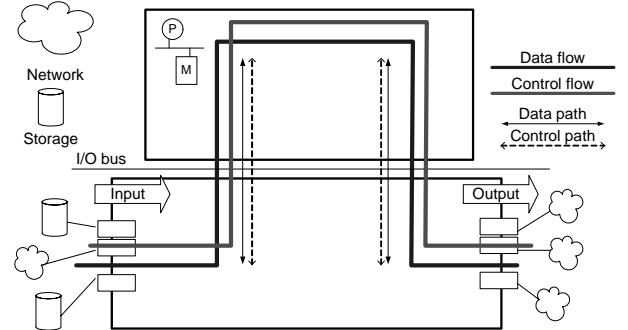


Figure 2 Edge server model with the current data flows

The distinct natures between control and data planes are well understood today [5]. The former demands arbitrary processing, while the latter demands repetitive data operations. This property indicates that control planes should run on generic-purpose computers while data planes should run on application specific devices. Now, this separation design principle is widely used in high-speed routers today; however, it has not been applied to the edge servers to any degree recognizable. The new architecture described in this paper constitutes an effort in this direction.

In the following description, a flow is defined to be a set of packets with the following properties: (1) having the same destination IP address, and the same destination TCP or UDP port number, (2) having the same source IP address, and the same source TCP or UDP port number. Since this paper focuses on TCP and UDP packet processing, we will ignore processing of other types of packets in the discussion.

The new architecture is presented in Figure 3. The I/O bus is chosen as the logic line that separates control and data planes. Control plane packets are processed by a GPA module that uses the classic server architecture, while data plane packets that qualify for acceleration are processed by the switching module.

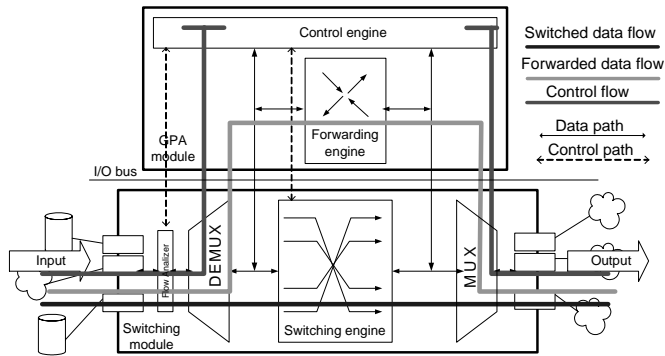


Figure 3 Separation of control plane and data plane in the new edge-server architecture

The new architecture differentiates among three types of packet flows:

- *Control flow*: any flow that carries packets on the control plane. These flows are processed by the GPA module.
- *Switched data flow*: any flow that carries packets on the data plane and that qualify to be processed by the switching module (the qualification criterion is further explained below). These flows are processed by the switching module.
- *Forwarded data flow*: any flow that carries packets that belong to the data plane and that do not qualify to be processed by the switching module. These flows are processed by the GPA module.

The following are the modules in the new architecture:

- *Flow analyzer and demultiplexer*. These are the first modules that any incoming packet finds upon arriving at the edge server. The flow analyzer classifies a packet based on the type of flow that it belongs to. Then depending on its flow type, the demultiplexer passes the packet to the control engine, the switching engine or the forwarding engine. This demux operation is described in Figure 4.
- *Control engine*. This module corresponds to the normal path in the current GPA. An example of a control engine is a software package that implements RTCP. A communication path between the control engine and the switching module is added so that control commands can be delivered from control plane to data plane (dotted lines in Figure 3).
- *Switching engine*. This module processes the data plane as fast as possible. To achieve high performance, the switching engine is implemented using a pipeline of microprocessors complemented with several ASIC engines that accelerate specific networking and storage operations. Via this engine, data are transferred from storage to network en mass.
- *Forwarding engine*. This module can also process data plane packets but at lower speeds, as it runs in the GPA module. Any data plane that has no performance requirements is handled by

this module. An example of a forwarding engine is the IP layer of an ordinary server.

- *Multiplexer*. This module multiplexes packets coming from the three different paths into the network interface.

The switching module is integrated in a DPA-1200, a full-length 64-bit PCI card powered by DPE-1000 DirectPath Engine (a Xiran product) that includes a Gigabit Ethernet port, SCSI and Fiber Channel ports. The flow analyzer, demultiplexer, switching engine and multiplexer are all integrated in the DPE-1000. The switching module can be connected to any video-streaming server equipped with a PCI slot. Such server acts as the GPA module as represented by Figure 3.

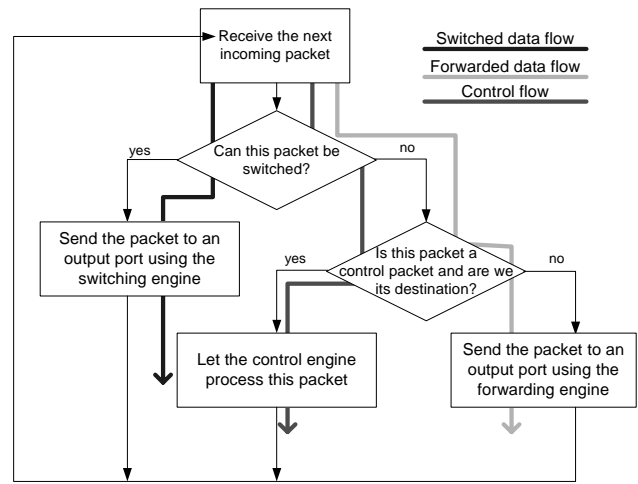


Figure 4 Demux operation

## B. Protocol plane-map

Several protocols can be used to stream video over the Internet. Some of them are proprietary (e.g. RDT from RealNetworks or MMS from Microsoft), and some are standardized by the IETF. While this architecture has been tested with most of these protocols, in what follows we use the example provided by the three streaming protocols standardized by IETF: Real Time Protocol (RTP), Real Time Control Protocol (RTCP) and Real Time Streaming Protocol (RTSP).

When used together, RTP, RTCP and RTSP define a suit of transport and application protocols necessary and sufficient to carry video streaming media. RTP is responsible to carry the actual video and audio packets. Some services provided by RTP include time reconstruction or loss detection. RTCP is responsible to control RTP. It provides functions such as QoS monitoring or source identification. RTSP is a separate control protocol used to provide VCR-like remote control functions, such as pause, fast forward, reverse, or absolute positioning.

Figure 5 shows how these protocols are mapped into the new architecture. Both RTCP and RTSP traffic operate in the control plane and, therefore, the flow analyzer routes them to the GPA module. On the other hand, RTP traffic, which belongs to the data plane and has high performance requirements, is routed to the switching module.

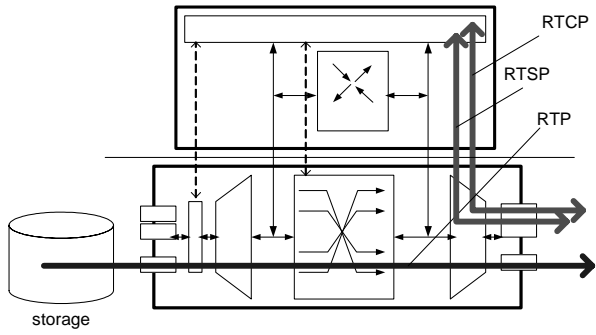


Figure 5 Protocol plane-map

### C. Performance scalability

In this section we review the six server overheads and show how they are resolved in the new architecture. Table 2 presents a summary of the techniques used to resolve them. Next, we also explain these techniques in more detail.

Table 2 Techniques in the new architecture that resolve the six overheads

Overhead	Technique	Implementation note
Extra data copies	Zero copy sockets.	The switched path, since it cannot be affected by malfunctioning applications, never performs the safety copy.
MTU inefficiencies	Hardware acceleration	ASIC engines performing TCP segmentation
Excessive I/O interrupts	Separation of control and data plane	The switched path never crosses the I/O bus
Excessive context switching	Network and storage stack awareness	An application aware scheduler that uses R2C is implemented
Expensive software operations	Hardware acceleration	ASIC engines performing checksum and encryption computations
Parallelization penalty	Network and storage stack awareness	A pipeline of application aware processors is implemented along the switched path achieving almost zero locking contention

1- *Zero copy.* In the new architecture, applications have no means to directly access the resources provided by the switching module and, therefore, the integrity of the latter is guaranteed. Because no further protection is required, the switching module is implemented using zero copy sockets. This means that, for instance, data coming from storage can now be transferred to the network via the switching module without incurring any data copy overhead.

2- *Hardware Acceleration.* Since the switching engine is implemented as a separate embedded device, hardware acceleration can be efficiently integrated, assisting computing

intensive operations. For example, the problem of MTU inefficiencies is solved by integrating an ASIC engine that performs TCP segmentation. Similarly, the problem of expensive software operations is solved by integrating ASIC engines that can perform operations such as hardware checksum or encryption.

3- *Separation of control plane and data plane.* The problem of excessive I/O interrupts is solved by a clean separation of control and data planes. By processing the data plane in the switching module, a complete movie can be transferred bypassing the I/O bus.

4- *Network and storage stack awareness.* Because the switching module is fully dedicated to the specific tasks of network and storage, its architecture can be designed accordingly to the requirements of the protocols. Content switching is removed by implementing a run-to-completion scheduler. Similarly, the locking overhead typically incurred in the generic purpose parallel architecture is resolved by distributing the tasks among the microprocessors using a pipeline approach.

## IV PERFORMANCE EVALUATION: TCP ACCELERATION

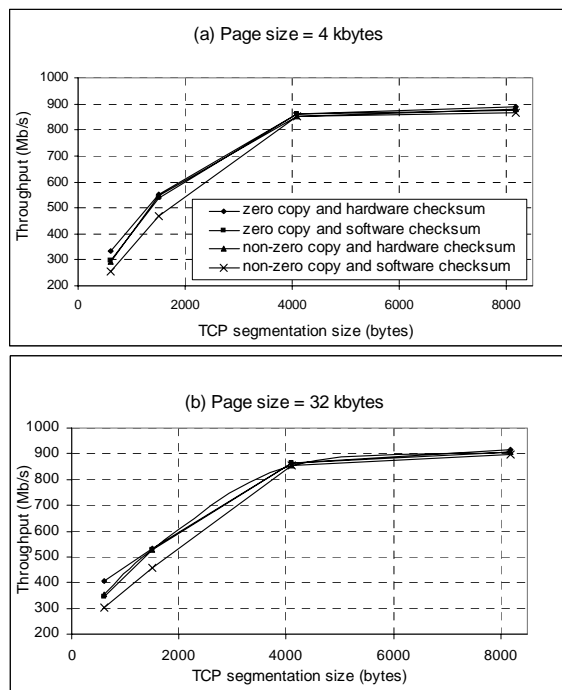
We evaluate the performance impact of three optimizations implemented at the switching engine: zero copy, hardware checksum, and hardware TCP segmentation. One switching module DPA-1200 is plugged into one of the PCI slots of a host PC server equipped with an Intel Xeon 2.4 GHz running Linux kernel 2.4.7-10. We transfer raw data from storage to network running the data plane on the switching module and the control plane on the Intel Xeon processor (GPA module) using HTTP protocol. Page sizes of 4 Kbytes (Figure 6a) and 32 Kbytes (Figure 6b) are used in these tests.

TCP segmentation improves the TCP throughput a 300% when the segment size increases from 600 bytes to 4096 bytes. Note that this increase is not linear due to the overhead of the small TCP acknowledgement packets, which cannot be accelerated through TCP segmentation. Both hardware checksum and zero copy have a larger impact for small sizes of TCP segmentation. This is due to a bottleneck shift in the pipeline architecture within the switching engine. Only when the segmentation size is relatively small, the microprocessors responsible for TCP become a bottleneck. Therefore, the real impact of hardware checksum and zero copy is found for small segmentation sizes, each one yielding a 15% improvement in throughput. Finally, it is worth noting that due to separation of data and control plane, in all the tests the CPU utilization of the Intel Xeon CPU was lower than 1%.

## V CONCLUSIONS

Ultimately, a system's superior performance can only be achieved by a logically efficient design. While the separation of control and data planes may appear to be obvious, its utility and usefulness are once again proved.

Even though the original Turing Machine did invoke the separation principle, the current server architecture has overlooked this obvious design principle to improve the content delivery capacity.



(\*) Host CPU utilization in all simulations is 1%

Figure 6 Evaluation of TCP performance

This paper represents a step towards the specialization way of system design. The problem of the edge server today is that its architecture subsumes general-purpose applications while the dominant applications are data related, due to the nature of the Internet. Specialization implies separate optimization of the dominant repetitive functions. The principle of separate

optimization is in fact the broader concept that leads to superior system performance. As Moore's Law implies a slower speed-up in processing than optical communications and storage capacity, it is all the more important to apply this principle to remove bottlenecks in processing.

#### ACKNOWLEDGMENTS

The work herein presented was first started in December 1999 by Irvine Networks, later becoming Xiran. This work is the effort of more than 30 hardware and software engineers that have worked together for the last 4 years developing DPA. This work is dedicated to all of them.

#### REFERENCES

- [1] John Hennessy and David Patterson, "Computer Architecture, a Quantitative Approach," 2<sup>nd</sup> Edition, Morgan Kauffman, 1996.
- [2] Marshall Kirk McKusick et al., "The Design and Implementation of the 4.4 BSD Operating System," Addison Wesley, 1996.
- [3] George Apostolopoulos et al., "Securing Electronic Commerce: Reducing the SSL Overhead," IEEE Network Magazine, July/August 2000 Vol. 14 No. 4.
- [4] Curt Schimmel, "Systems for Modern Architectures: Symmetric Multiprocessing and Caching for Kernel Programmers," Addison Wesley Professional, 1994.
- [5] L. Yang, et al., "Forwarding and Control Element Separation (ForCES) Framework," Internet Draft, Working Group ForCES, October 2003.
- [6] J. Mogul, "TCP offload is a dumb idea whose time has come", 9th Workshop on Hot Topics in Operating Systems, USENIX, Lihue, USA, May 2003.
- [7] E. M. Nahum, et al., "Performance issues in parallelized network protocols," USENIX Symposium on Operating Systems Design and Implementation, Monterrey, USA, November 1994.