◆ A Novel Network Processor for Security Applications in High-Speed Data Networks

Kyriakos G. Vlachos

This paper describes the programmable protocol processor (PRO3) architecture, which is capable of supporting advanced security services over high-speed networks. Security services include such things as a firewall, packet and flow classification, connection-state handling (i.e., stateful inspection), higher-layer protocol data unit (PDU) reassembly (i.e., application-level firewalls), and packet encryption and decryption. The PRO3, which is integrated with a high-speed line card, attempts to accelerate the performance of the firewall by implementing key functionality in hardware and by optimizing the balance between hardware and software functions. In this way, significant performance enhancements can be achieved, such as making transport control protocol (TCP) and Internet protocol (IP) data transactions secure, and protecting and separating virtual private networks (VPNs) from the external public network. The PRO3 incorporates an innovative scheme—a reduced instruction set computing (RISC)-based pipelined module with line-rate throughput—that makes it possible to process high- and low-level streaming operations efficiently. Using microcode profiling and simulation, we give performance results for a stateful-inspection firewall application with network address translation (NAT) support. © 2003 Lucent Technologies Inc.

Introduction

Rapid advances in optical networking technology have increased the capacity of physical interconnection links to such a point that bandwidth can be considered an available resource. As a result, the network bottleneck has shifted from transport to the network nodes, especially those (at the edge of the core and access networks) that must keep pace with increasing line rates. Traditional, software-based network processing is no longer adequate, because the average time it takes to serve a single packet can be orders of magnitude larger than the average time between the reception of sequential packets, so line-rate processing cannot be sustained [4, 14]. Gradually, next-generation telecommunications systems are rendering legacy software-based and generic microprocessor-based systems insufficient for network processing [1, 16].

One of the main responsibilities of these network processing units is to provide firewall functionality, by granting access to users in a protected fashion, and by separating a company's public server (e.g., its Web server) from its internal private network [25]. Until recently, this type of firewall system was mainly software-based, and ran on general-purpose

Bell Labs Technical Journal 8(1), 131–149 (2003) © 2003 Lucent Technologies Inc. Published by Wiley Periodicals, Inc.
Published online in Wiley InterScience (www.interscience.wiley.com). • DOI: 10.1002/bltj.10058

processors; the actual firewall was built in software for a particular processor [11, 15]. However, as the number of concurrently active connections increases, the number of protocol instances and the workload on the system processor also increases. Currently, the traffic load is such that a software-based firewall system is no longer adequate for high-speed networks, because it cannot handle the traffic efficiently.

A further problem is that, in order to reach control decisions (e.g., whether to accept, reject, authenticate, encrypt, or log communication attempts) for transport control protocol (TCP)- and Internet protocol (IP)-based services, a firewall must retrieve, store, and manipulate protocol data from all the network layers of the open systems interconnection model. For this to be possible, services such as packet and flow classification, connection-state handling (i.e., stateful inspection), higher-layer protocol data unit (PDU) reassembly, and packet encryption and decryption must be included in a security system [13] and—more important—must be executed in less time than the average time between the reception of sequential packets [3].

A solution to these network-processing problems is not immediately evident. Line rates will continue to increase, imposing ever-new processing requirements [2, 14]. Furthermore, particularly in the area of advanced firewalls and security systems [6, 7], the demand for more sophisticated and complex functions (e.g., high-speed stateful packet inspection [22]) is increasing. Application-specific integrated circuit-based systems require extended periods of development—12 to 18 months from design to marketplace—and, although very efficient, do not provide the necessary flexibility. A system based on a highly programmable reduced instruction set computing (RISC) core can provide flexibility, but it sacrifices speed to programmability. One way to achieve both packet-handling speed and programming flexibility is to adopt a hybrid solution in which RISC cores are integrated with dedicated hardware [24]. So-called network processors (NPs) [9, 19], which do this, offer increased throughput and low latency in a broad range of applications by allowing networking tasks normally handled by software to be executed by hardware. A network processor can be defined as a highly integrated communications circuit that is optimized to provide programmable processing of PDUs at high—preferably, at wire—speed.

Panel 1. Abbreviations, Acronyms, and Terms

AAL—ATM adaptation layer
ATM—asynchronous transfer mode
BGA—ball grid array
CAM—content-addressable memory
CMOS—complementary metal-oxide semiconductor
CPU—central processing unit
CRC—cyclic redundancy check
DMM—data memory manager
DRAM—dynamic RAM
DSP—digital signal processor
EDN—Electronic Design News
EEMBC—EDN Embedded Microprocessor Benchmark Consortium
FEX—field extractor
FIFO—first in first out
FMO—field modifier
FSM—finite state machine
ICMP—Internet control message protocol
IP—Internet protocol
MPLS—multiprotocol label switching
NAT—network address translation
NP—network processor
PDU—protocol data unit
PPE—protocol-processing engine
PRO3—programmable protocol processor
QoS—quality of service
RAM—random access memory
RISC—reduced instruction set computing
RPG—RPM glue logic
RPM—RISC-based pipelined module
RSVP—resource reservation protocol
SAR—segmentation and reassembly
SRAM—synchronous RAM
TCP—transport control protocol
TOS—type of service
TRS—traffic scheduler
TSC—task scheduler
TTL—time to live
UDP—user datagram protocol
VLSI—very large scale integration
VCI—virtual channel identifier
VPI—virtual path identifier
VPN—virtual private network

In this paper, we describe the programmable protocol processor (PRO3) system architecture [20], which is a hybrid approach to the challenging protocol-processing problem. It has been designed to support and accelerate the performance of a stateful-inspection firewall with network address translation (NAT) support [8] by using tight coupling of software and hardware to produce more efficient execution of telecommunications protocols. The PRO3 architecture is based on a high-performance RISC core, which is extended with programmable, pipelined hardware. Real-time protocol functions and functions that place heavy demands on the central processing unit (CPU) are handled by the programmable hardware; the remaining functions—as well as higher-layer protocols—are handled by a pair of on-chip RISC-processor-based modules optimized for fast context-switching and pipelined processing [17]. With this architecture, it is expected that significant performance improvements will be achieved in:

- The rate of connection insertion and deletion, which measures the number of connections per second supported by the system when applications set up and tear down connections continuously;
- The throughput, which measures the aggregate number of bytes per packet that the system can process and forward to its output interface;
- The latency, which measures the aggregate delay encountered by network traffic, and which is introduced by the processing delay of the system; and
- The number of processed sessions, which measures the maximum number of simultaneous connections supported by the system.

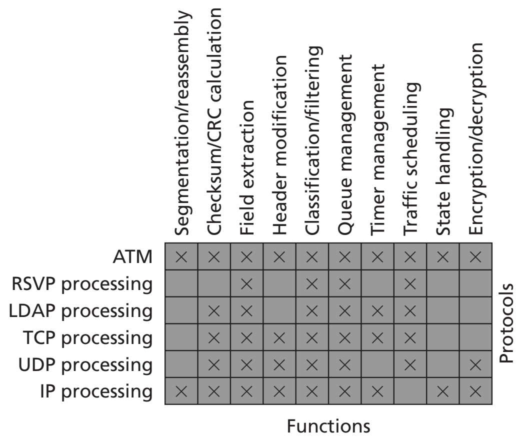
The rest of this paper is organized as follows. The next section analyzes the basic functions required in a high-speed firewall system and discusses their implementation by the PRO3 architecture. The following section describes the PRO3 architecture; the subsequent section analyzes the implementation of and provides a performance evaluation of a stateful-inspection firewall on the PRO3 system. It also gives performance simulation results and measures the cycle budget for each critical processing module. A final section draws conclusions regarding the overall performance of the PRO3 system, assuming worst-case throughput by its submodules.

Functions and Requirements of an NP-Based Firewall System

Based on the protocol analysis presented in [23], **Figure 1** summarizes the required functionality found in various telecommunications systems that involve protocol processing. The following subsections analyze each function displayed in Figure 1 and discuss how the PRO3 system implements it.

Field Extraction

The packet header contains all the information necessary for processing asynchronous transfer mode (ATM) cells or IP packets. Because the entire packet travels as an aggregation of bits, a mechanism is needed to separate header fields from the rest of the message as well as from each other. Such a process has to be adaptable to the protocol in question and take into account the length (variable or not) of the fields to be extracted. Because header fields may vary


Protocols
Functions

	Segmentation/reassembly	Checksum/CRC calculation	Field extraction	Header modification	Classification/filtering	Queue management	Timer management	Traffic scheduling	State handling	Encryption/decryption
ATM	×	×	×	×	×	×	×	×	×	×
RSVP processing			×		×	×		×		
LDAP processing		×	×		×	×	×	×		
TCP processing		×	×	×	×	×	×	×		
UDP processing		×	×	×	×	×		×		×
IP processing	×	×	×	×	×	×	×		×	×

ATM—Asynchronous transfer mode
CRC—Cyclic redundancy check
IP—Internet protocol
LDAP—Lightweight directory access protocol
RSVP—Resource reservation protocol
TCP—Transport control protocol
UDP—User datagram protocol

Figure 1.
Protocols and required functionality.

in length, their boundaries are not fixed. Therefore, the field extraction mechanism takes the form of a variable-width window that slides over the message header, and it has to deal with the problems of positioning and size adjustment.

PRO3 provides programmable field extraction units that support the processing of variable-length IP packets, TCP and user datagram protocol (UDP) packets, fixed-size ATM cells, and variable-length ATM adaptation layer (AAL) PDUs.

Segmentation and Reassembly

In network protocols, it is quite common for packets to be fragmented before they are issued to the network, because there are restrictions regarding the maximum number of bytes that can be transferred over a specific data-link interface. When fragmentation occurs, each one of the fragments (or segments) created from the byte stream is marked to identify its position within the stream. Then, when the fragments reach their destination, a reverse procedure—reassembly—is followed. During the reassembly procedure, fragments may be received out of order; if this occurs, special care must be taken to reorder them.

PRO3 provides reassembly queues in the buffer memory (which is used to store packets temporarily after they are received) for the reassembly of either IP packets or AAL-layer PDUs.

Checksum and CRC Calculation

Checksum and cyclic redundancy check (CRC) calculations are performed to verify that the information carried in the header and user data have been transmitted correctly. If an error is detected, the packet containing the error is discarded. In the IP world, the checksum is calculated as the 16-bit one's complement of the one's complement sum of all 16-bit words in the protected area. For purposes of computing the checksum, the value of the checksum field is zero. For the IP protocol, the checksum protects only the IP header, but, for the TCP and the UDP, both the header and the user data are protected.

PRO3 provides dedicated hardware blocks to accelerate the data verification algorithm in the most common cases, which include IP checksum and 10- and 32-bit CRC calculations.

Header Modification

Header modification updates the values of specific fields of a protocol header after the classification procedure has been performed. Fields that may be updated include:

- The TTL field of the IP header,
- The TOS byte of the IP header,
- The header checksum and CRC,
- The IP address and port in NAT,
- The next-hop virtual path identifier (VPI) and virtual channel identifier (VCI) values of an ATM cell, and
- The next-hop medium access controller address (which may be inserted when the packet is being sent to an Ethernet output port).

PRO3 supports both header modification and PDU encapsulation functions.

Classification and Filtering

Classification maps an incoming cell or packet to a homogeneous class of cells or packets. All the cells and packets of a specific class share some common attributes, among them quality of service (QoS) parameters, routing path, and session and connection parameters. Classification requires a look-up in a table that associates several combinations of values appearing in fields of protocol-specific information with a classification decision. The classification of packets based on higher-layer protocol information in intermediate nodes also requires the reassembly of packets in each node.

PRO3 uses a ternary content-addressable memory (CAM) to provide flow classification capabilities. After classification, traffic is filtered; only data that comply with the network policy pass through the network processor. For IP-based applications, connection endpoints are identified by a 5-tuple: the IP source address, the IP destination address, the source port, the destination port, and the protocol type (i.e., TCP or UDP). For ATM applications, connection identification is based upon the input port, the VPI and VCI fields, and (for signaling applications) higher-layer connection identifiers. PRO3 supports both external configuration and connection-parameter initialization.

Queue Management

The main use of queues in network protocols is to buffer incoming cells and packets temporarily, until they are fully processed and ready to be forwarded. Queues can also be used to store maintenance and state information for handling incoming cells and packets. A queue manager must, at the very least, support an insert and extract interface. PRO3 provides adequate buffer space for storing and reassembling packets for further processing.

Timer Management

Timeout timers and watchdog timers are an important part of network protocols. Because each protocol instance needs at least a few timers (i.e., one for each different time scale), a system supporting thousands of protocol instances must devote a considerable part of its processing time simply to keeping track of the time for all the timers.

PRO3 incorporates dedicated hardware that can handle a timer pool large enough to support the requirements related to timer-event generation of the application described in the section "Implementation and Performance Evaluation of a PRO3-Based Stateful-Inspection Firewall."

State Handling

State handling refers to the maintenance (i.e., the creation, update, and deletion) of protocol state information. State information must be organized in such a way that it will allow (O(1)) searches fast enough to achieve the required throughput. The execution of the protocol finite state machine (FSM) is supported by microcode execution on the PRO3 RISC CPU and by an efficient context-switching mechanism.

Traffic Scheduling

Traffic scheduling involves the forwarding of different packet streams, using a set of queues and, possibly, other mechanisms like timers. PRO3 supports only generic scheduling based on priority queues for sharing PRO3 processing resources and resolving contention among packets to be transferred to the output interface.

Encryption and Decryption

Encryption usually occurs either before data are issued to the network or before data leave the local intranet for the Internet; decryption occurs either when data either arrive at their destination or when data enter the corporate intranet. Encryption and decryption are useful when dealing with virtual private networks (VPNs).

The integrated RISC and digital signal processor (DSP) of PRO3 supports the processing of encrypted packets for a number of flows. However, the throughput for encrypted flows will be lower than the link rate.

The PRO3 Architecture

The PRO3 system architecture follows a unique approach to high-speed protocol processing. The protocol processor attempts to accelerate the execution of telecommunications protocols by extending a high-performance RISC core with programmable, pipelined hardware. The PRO3 system offers three end-to-end processing paths:

- Hardwire packet reception, storage, and forwarding;
- Wire-speed packet processing, involving specialized RISC cores; and
- Best-effort processing, involving typical RISCs.

Real-time protocol functions and functions that place heavy demands on the CPU are handled by the programmable hardware; the remaining functions—as well as higher-layer protocols—are handled by the on-chip RISC in an integrated way or by the external processor.

The innovative concept of the PRO3 architecture is to provide the processing power required for high-speed protocol processing by incorporating parallelism and pipelining and—wherever possible—integrating generic microprogrammed engines with hardware components dedicated to performing specific protocol-processing tasks that rarely change over time, such as framing, CRC and checksum calculation, hashing, timer management, flow management, and memory management. Such hardware has only a few configuration options, and does not require the support of programmable software.

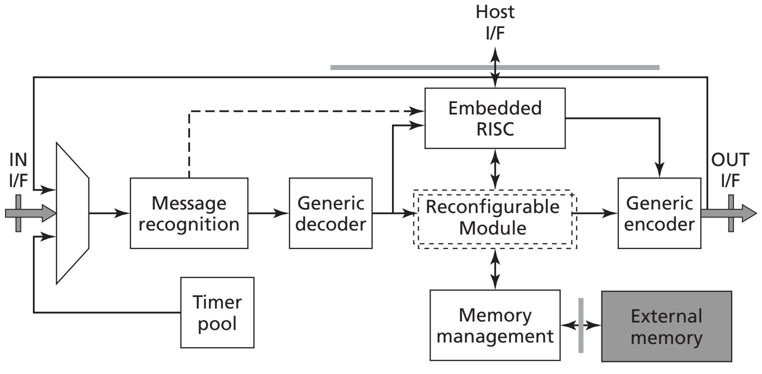
The PRO3 architecture employs specialized cores to handle high-speed links and support demanding applications. This is necessary, because some protocol-processing tasks are heavy users of computer resources, either because of their computational complexity or because of their demands on memory throughput. After analyzing the functions required for firewall functionality and determining how often they are called [23], it is possible to accelerate the critical ones significantly by using either fixed (for well-defined functions that are standardized) or programmable hardware. This approach allows a single component with different configurations to execute many different protocol FSMs that require both high execution and the ability to handle messages with low propagation and processing delay.

Logical Decomposition

The functional architecture of the protocol processor is depicted in **Figure 2**. The main concept behind the architecture is the idea of dividing protocol processing functions into tasks that can be executed in a distributed manner on different functional entities. The PRO3 functional model consists of a CPU with an embedded RISC core, a reconfigurable module, and a set of on-chip peripherals that are common to protocols and streaming tasks. The on-chip peripherals of the protocol processor contain the following modules: message recognition, generic encoder, genetic decoder, timer pool, and memory management.

The functional flow is similar to that followed when a protocol is executed in a typical software implementation. After a message or packet has been recognized and classified, it is assigned a flow ID that is unique within the respective protocol or task. Then, a field decoder extracts the necessary fields and control information and forwards structured information to the processing unit. Finally, a field or packet encoder composes the outgoing byte stream or packet.

Generally, protocol processing is initiated by the arrival at the network interface of a packet with specific protocol information in an appended header or trailer. The proper evaluation of the fields that contain the protocol information leads to the appropriate classification of the message. The reconfigurable module handles the execution either of entire protocols or of the most frequently used and most time-consuming branches of protocol FSMs in error-free conditions, depending upon the requirements of the application, the type of the message, and the protocol executed. The reconfigurable module is accessible to the main RISC CPU, in which configuration code is executed and protocol state information exchanged. In most cases, the result of protocol processing is an update of the stored protocol state and the generation—by the generic encoder and decoder—of a new or modified message or packet to be forwarded to either a higher-layer protocol or an output network interface. The symbolic feedback bus in Figure 2 indicates the return of messages to the input of the component, which occurs when a multiprotocol stack (e.g., IP and



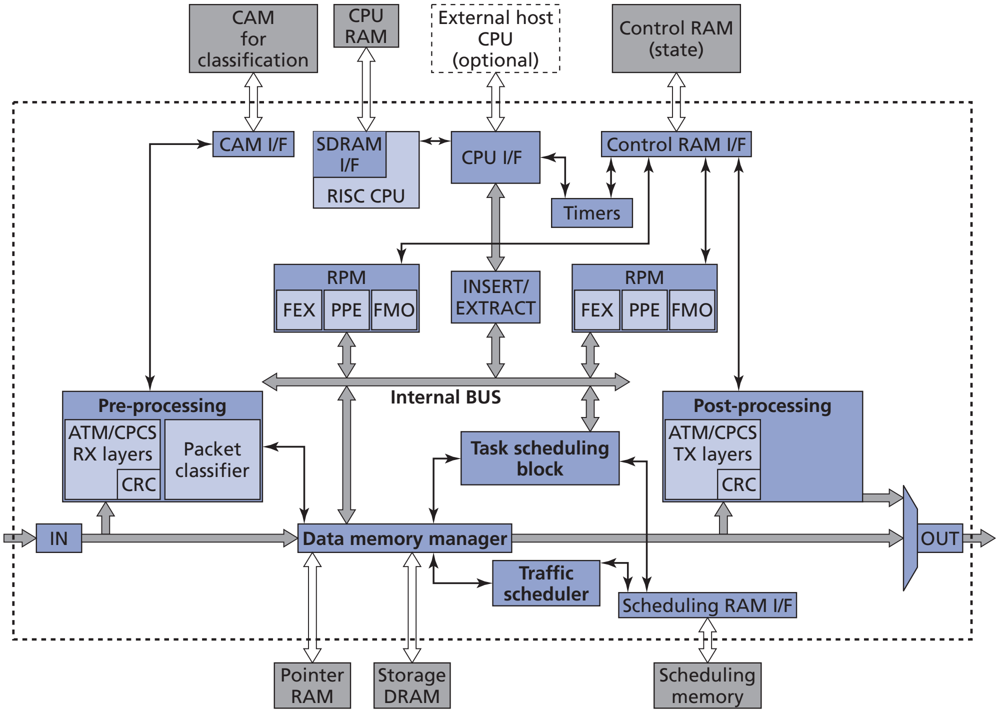
I/F—Interface
PRO3—Programmable protocol processor
RISC—Reduced instruction set computing

Figure 2.
PRO3 functional architecture.

ATM, or multiprotocol label switching [MPLS]) is implemented. Certain priority rules can be applied, along with specific access to or from the system. Timers and efficient memory management (including table look-up and data and protocol context buffering) are integral parts of protocol processing and potential bottlenecks in generic architectures, but in the PRO3 architecture they can be assigned to dedicated hardware units, as shown in Figure 2.

Physical Decomposition

The physical architecture of the PRO3 is depicted in **Figure 3**. In this figure, a dotted line separates internal from external blocks. The PRO3 has been designed as a system-on-chip, integrating hardware blocks for preprocessing and post-processing. It consists of two RISC-based pipelined modules (RPMs) (each consisting of two RISC-like cores for field processing and an optimized typical RISC [10] for structured processing), a header processor for programmable classification, and a typical RISC core for control and glue logic. Packet preprocessing and lower-layer protocol functions are executed by means of hardwired functionality (like the full ATM/common part convergence sublayer layers), and programmable PDU processing and packet classification by means of a RISC-like microengine for field extraction and a



ATM—Asynchronous transfer mode
CAM—Content-addressable memory
CPCS—Common part convergence sublayer
CPU—Central processing unit
CRC—Cyclic redundancy check
DRAM—Dynamic RAM
FEX—Field extractor
FMO—Field modifier

I/F—Interface
PPE—Protocol-processing engine
RAM—Random access memory
RISC—Reduced instruction set computing
RPM—RISC-based pipelined module
RX—Receive
SDRAM—Synchronized RAM
TX—Transmit

Figure 3.
Block architecture of PRO3 system.

controller of a high-throughput external ternary CAM device. The ternary CAM is used for flexible and deterministic classification. For IP packet processing, the packet classifier submodule constructs a search key of up to 144 bits that triggers the CAM search. This allows the PRO3 to be applied to other networking protocols, including MPLS, IPv6, and gigabit Ethernet. Memory latency bottlenecks caused by worst-case conditions have been avoided by allowing parallel memory accesses per functional unit.

Basic components in the PRO3 architecture are the RPM; the data memory manager (DMM), which stores the incoming traffic and, when requested, retrieves some or all of the stored packets; and a composite scheduler unit, which determines the internal data transaction and shapes the traffic based on priority specifications, and a packet classifier, which is used for packet classification and filtering. Altogether, the PRO3 chip embeds five RISC-like cores optimized for field processing, three typical RISC cores for packet processing, and 11 generic and application-specific hardware blocks. These are all implemented in UMC 0.18-micron complementary metal-oxide semiconductor (CMOS) technology occupying about 52 square millimeters, and are prototyped as a 1096 BGA package at 200 MHz. The following sections analyze the basic components of the PRO3 system.

The data memory manager. The main function of the DMM is to store incoming packets in and retrieve packets from the data memory. Packets are stored per flow in an external dynamic random access memory (DRAM) in queues implemented as linked-list data structures [18]. Each flow is served by the DMM through its dedicated queue (there is one queue per active connection in which data and packets are stored and reassembled), which is directly indexed by a flow ID value assigned by the packet classifier of the preprocessing block. This value uniquely identifies the protocol data and context for each connection and each layer of the protocol stack. The DMM shapes the incoming packets into fixed-size segments of 64 bytes. This segmentation of memory space optimizes the use of memory, improving the performance of the DMM and reducing the delay experienced by high-priority packets. In response to commands, the DMM retrieves from each incoming TCP/IP packet only the first 64 or 128 bytes (depending on the packet length), which contain all the header information. These segments are sent over the internal bus to the RPM modules, or to the control RISC CPU, or to a host CPU (via the insert and extract interface), or directly to the output interface. The remaining packets are stored in the data memory until the header information is fully processed, at which point they are ready to be transmitted or discarded.

This efficient way of storing and retrieving packets minimizes demands on memory throughput, giving the architecture a clear advantage over other network processor designs. The total bandwidth of the DMM is 10 Gb/s; it is dynamically distributed to four ports, which are used for receiving traffic from and transmitting traffic to the internal PRO3 bus and the network, respectively. The DMM can perform per flow queuing for up to 512K flows, and it operates at both the cell and the packet level, making it suitable for both cell- and packet-based applications. The DMM uses an external double data-rate DRAM for packet storage and a synchronous random access memory (SRAM) for segment and packet pointer storage.

The RISC-based pipelined module. The RPM consists of three logical units: the field extractor (FEX) programmable engine, the protocol-processing engine (PPE) (which is a composite module), and the field modifier (FMO) programmable engine. The PPE itself consists of three modules: a modified Hyperstone RISC [10], the RPM glue logic (RPG), and the read/write control RAM module. **Figure 4** displays the RPM top-level design architecture.

Together, the three modules of the PPE form a powerful three-stage pipelined module that contains the hardware and software required to form the processing heart of the system. This design, in which dedicated functional units are interconnected with a RISC core, is very well suited to tasks involving a high degree of functional diversity. The design also increases the use and the efficiency of the optimized RISC processor core by providing the means to configure its circuits for special tasks and, conversely, the RISC processor core—with its highly optimized

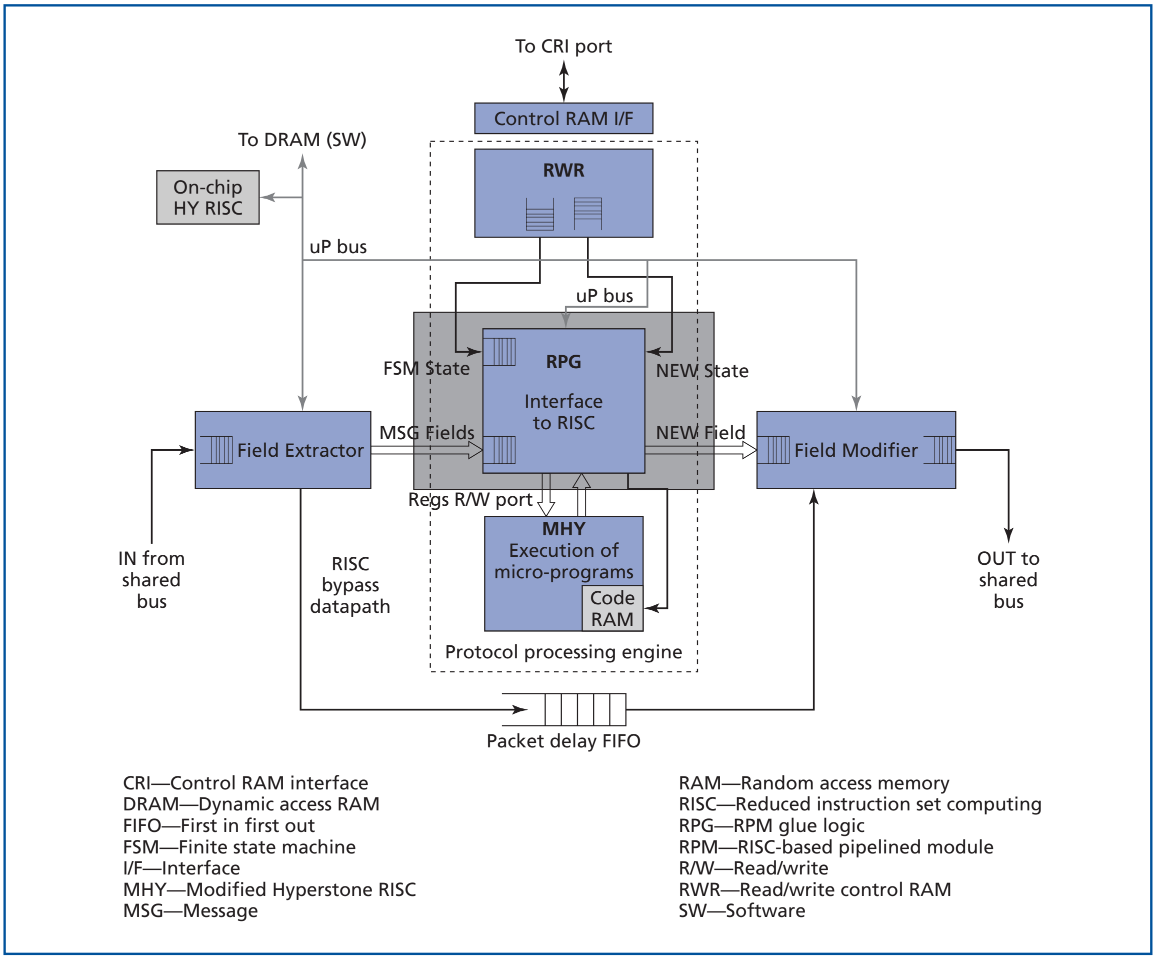


Figure 4.
RISC-based pipelined module.

configuration—accelerates protocol processing (and any other computing task), providing better performance at a lower cost than other network processors. The RPG block receives extracted fields from the FEX, and directly accesses the register file of the modified RISC core. This register file is divided into two parts: while the modified RISC core processes the data of one part, the RPG writes new fields to or reads new fields from the other part. A feedback signal from the RISC core to the entire RPM can extend the processing cycle of any given packet. This signal stalls the operation of the RPM, making it able to accommodate requirements that call for extended packet processing.

An important component of the RPM architecture is the packet delay first in first out (FIFO). After the FEX has extracted fields, the data received—a packet or a part of a packet—is temporarily stored in the packet delay FIFO to await the results of the processing of the fields or packet. Then, when the FMO receives the results of the processing and the delayed data from the RISC, it may, depending upon the processing results and the application, modify the delayed packet and send it back to the DMM. The DMM then replaces the segments of the packet that have been sent to the PPE by these modified data, and sends the new packet out. In other words, only specific fields are extracted by the FEX and fed to the PPE module, and only these specific fields are replaced by their new values in the FMO. This results in a constant clock-cycle-to-packet-length ratio and in an optimal total processing time, as measured against packet reception time.

The FEX and FMO engines of the RPM module are pipelined and fully programmable engines that operate on protocol-based firmware. Their operation is controlled by microcode stored in an internal SRAM with up to 2K instructions. The instruction set comprises simple and generic (i.e., protocol-independent) instructions that can be used in any protocol or protocol encapsulation scheme. **Table I** lists the fields that are extracted for processing from the IP, TCP, UDP, and Internet control message protocol (ICMP) headers during protocol processing in a stateful-inspection firewall application with NAT support.

Table I. Fields extracted for protocol processing in a RISC-based pipelined module.

Fields Extracted (RPM module)			
Header	Field	Description	Number of bits
IP	ip_hlen	IP header length	4
	ip_len	IP packet length	16
TCP	th_sqn	TCP sequence number	32
	th_ack	TCP ack/ment number	32
	th_off	TCP offset	4
	th_flags	TCP flags	6
	th_win	TCP window size	16
	th_cksum	TCP checksum	16
UDP	No field is extracted, because it is stateless.		
ICMP	No field is extracted, because it is stateless.		

ICMP—Internet control message protocol
IP—Internet protocol
RISC—Reduced instruction set computing
RPM—RISC-based pipelined module
TCP—Transport control protocol
UDP—User datagram protocol

The packet classifier. Classification is an important and complex operation in protocol processing environments. The packet classifier is incorporated in the preprocessing block; it is realized using dedicated programmable logic in conjunction with a ternary CAM for fast pattern matching. The packet classifier module receives packet data from the network interface (i.e., packet over synchronous optical network) and performs protocol header verification, field extraction, flow classification, and exception handling (e.g., inserting and deleting flows). The packet classifier module consists of three submodules: a FEX engine, a classifier FSM, and a verifier. The purpose of the field extraction mechanism is to extract—actually to isolate—specific fields in a packet, and to forward them to the classifier FSM for processing. These fields can be part of the packet header or trailer (or packet headers and trailers, in the case of packet encapsulation). The FEX engine is similar to the one incorporated in the RPM module. The classifier FSM constructs a 144-bit search key that triggers a CAM search. In response, the CAM returns a 19-bit flow ID value.

The scheduler unit. Scheduling is necessary to resolve contention for processing resources in a fair manner and to distribute over time the transmission of packets and cells (in a network medium) according to traffic management rules (i.e., traffic shaping). The meaning of fairness varies depending upon the metric under consideration, but it may imply low latency or a guaranteed share of the resources. When the processor cannot sustain worst-case conditions with line rates of 2.5 Gb/s or 10 Gb/s (i.e., the standard line rates for TCP stateful inspection), queuing is necessary, and a queuing service discipline that guarantees QoS must be implemented. Therefore, the scheduler unit must maintain a number of priority queues, in order to schedule the forwarding of packets for processing according to a configurable priority per flow or per QoS class. In PRO3, the scheduling mechanism has been divided into two parts: one for task scheduling and one for traffic scheduling. The task scheduler (TSC) block (see Figure 3) controls the data flow in the high-speed internal PRO3 bus, which is used for data transactions and communication among the internal modules. The traffic scheduler (TRS) block (see Figure 3) manages the profile of the generated traffic in accordance with traffic management specifications and service-level agreements. The TRS block is of paramount importance in implementing and guaranteeing an agreed-upon QoS.

Together, the TSC and the TRS manage 32 scheduling queues that can be used for sharing PRO3 processing resources in a weighted round robin manner [12]. Each of these queues is associated with one of the possible internal destinations for packets in PRO3 and with a specific handler protocol that will be executed for the data of this flow. Naturally, more than one data queue will share the same scheduling queue. The multiplexing of multiple flows in one scheduling queue (i.e., flow group) is based on a round robin discipline. Thus, all the flows that hash into the same scheduling queue will share equally the portion of internal processing resources (in terms of service opportunities) that is allocated to that queue in accordance with the preconfigured weight of the queue.

Implementation and Performance Evaluation of a PRO3-Based Stateful-Inspection Firewall

The proposed firewall implementation on the PRO3 system is a departure from the usual implementation of firewalls that attempts to increase both the effective bandwidth and the number of concurrently monitored connections. In order to provide robust security, a firewall must continuously monitor the line data, and must track and control the flow of all communication passing through the firewall. In this implementation, the PRO3-based firewall is positioned at the network boundaries on an edge router line card. To reach control decisions for TCP/IP-based services (i.e., to decide whether to accept or reject a packet), the PRO3 system stores, retrieves, and processes protocol data derived from different communication and protocol layers. Moreover, since it is not sufficient to examine packets in isolation, state information is maintained and is used for making control decisions. State information—derived either from the recent past of a communication session or from other applications run in the past—is an essential factor in making control decisions for new communication attempts. Depending upon the communication attempt, both the communication state (derived from past communications) and the application state (derived from other applications) may significantly affect the control decision. By maintaining state information, the PRO3-based stateful-inspection firewall can meet all the security requirements of a firewall; traditional firewall technologies (e.g., packet filters and application-layer gateways) cannot. Stateful inspection takes place mainly at the boundary between the intranet and the Internet. The PRO3 system accelerates the performance of the firewall by implementing key functionality in hardware, as well as by optimizing the balance between hardware and software functions.

The mapping of the firewall application to the PRO3 architecture is displayed in **Figure 5**. The PRO3-based stateful inspection firewall has the following features:

- It can accept and forward packets belonging to certain flows (i.e., TCP, UDP, and ICMP) without stateful-inspection processing of protocol headers.
- It can accept and forward packets belonging to certain flows (i.e., TCP and ICMP) that respect the rules of stateful-inspection processing.
- It can reject packets belonging to certain flows (i.e., TCP, UDP, and ICMP).
- It can reject packets belonging to certain flows (i.e., TCP, UDP, and ICMP) that violate the rules of stateful-inspection processing.
- It can provide NAT service to packets that belong to certain flows (i.e., TCP and UDP).

The stateful-inspection firewall for TCP packets consists of three main stages. (The internal data path of each stage is displayed in **Figure 6**.) They are:

- *Packet reception*. The input module (the IN block in Figure 6) receives flows of IP packets and forwards them to the packet classifier in the preprocessing block and to the DMM. The packet classifier makes a preliminary check, performs field extraction and classification, and sends

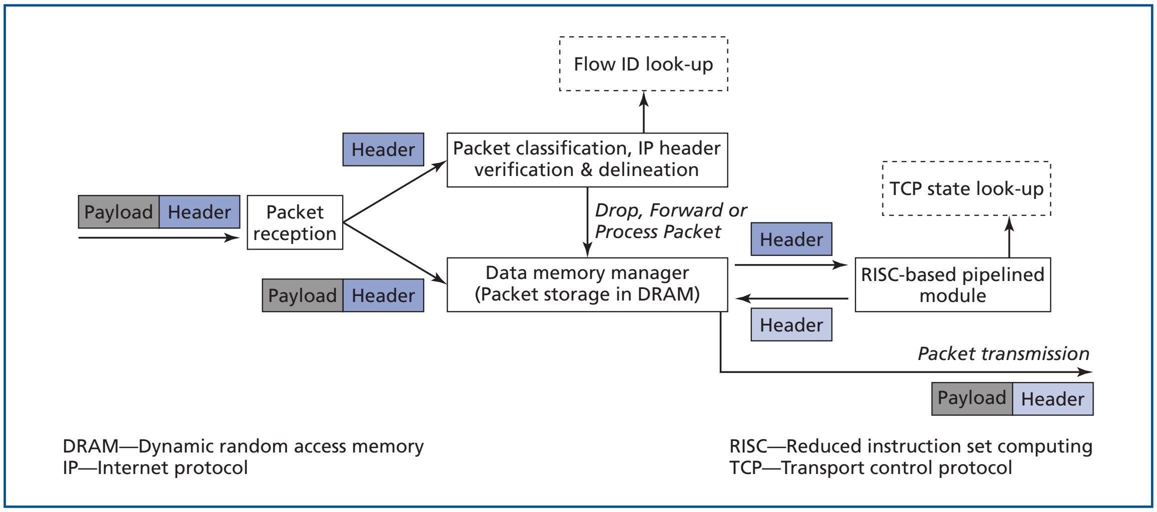

Flow ID look-up
Header
Packet classification, IP header verification & delineation
Payload
Header
Packet reception
Drop, Forward or Process Packet
TCP state look-up
Header
Data memory manager (Packet storage in DRAM)
RISC-based pipelined module
Payload
Header
Header
Packet transmission
Payload
Header
DRAM—Dynamic random access memory
IP—Internet protocol
RISC—Reduced instruction set computing
TCP—Transport control protocol

Figure 5.
Mapping of the firewall application on the PRO3 architecture.

control information to the DMM. The DMM segments the IP packet, stores it in a temporary queue, and calculates and stores the packet length. As soon as the DMM receives all the relevant information from the packet classifier, it appends the packet to the appropriate queue and sends control information to the schedulers.

- *Packet processing*. In this stage, certain fields are extracted from the protocol headers and brought into the RPM along with the relevant state information, which is stored in the control RAM. The control RAM is a zero-bus-turnaround SRAM, which is accessed by multiple blocks requesting flow-state information at various times. All this information is processed based on the stateful-inspection processing rules, and a decision is made to accept or to reject the current packet. In necessary, the NAT service is also activated.
- *Packet transmission*. The transmission operation is based on a forward-upon-availability principle. The traffic forwarded to the network keeps the traffic profile of the incoming streams. This is accomplished by having the task scheduler preserve the order of the packets it processes, and by having it support different priorities for internal processing.

Performance Evaluation

It is worth noting that there is no simple way of doing a performance evaluation for a protocol processor. Because protocol processors constitute a new paradigm in network-oriented computing architectures, few benchmark results exist. Furthermore, no standard benchmarking procedures exist, because of the variety of protocol processor architectures and the wide range of applications. Our approach to evaluation combines legacy benchmarking metrics for estimating the performance of programmable microengines (e.g., instructions per second and instructions per cycle) with the techniques of the NP Benchmarking Working Group of the Electronic Design News (EDN) Embedded Microprocessor Benchmark Consortium (EEMBC), based on its first published draft [5]. A new metric introduced by the EEMBC is headroom, which measures the ability of a network processing platform to perform multiple networking functions in parallel, in any combinations that make sense for networking applications, and still maintain wire-speed performance. In the PRO3

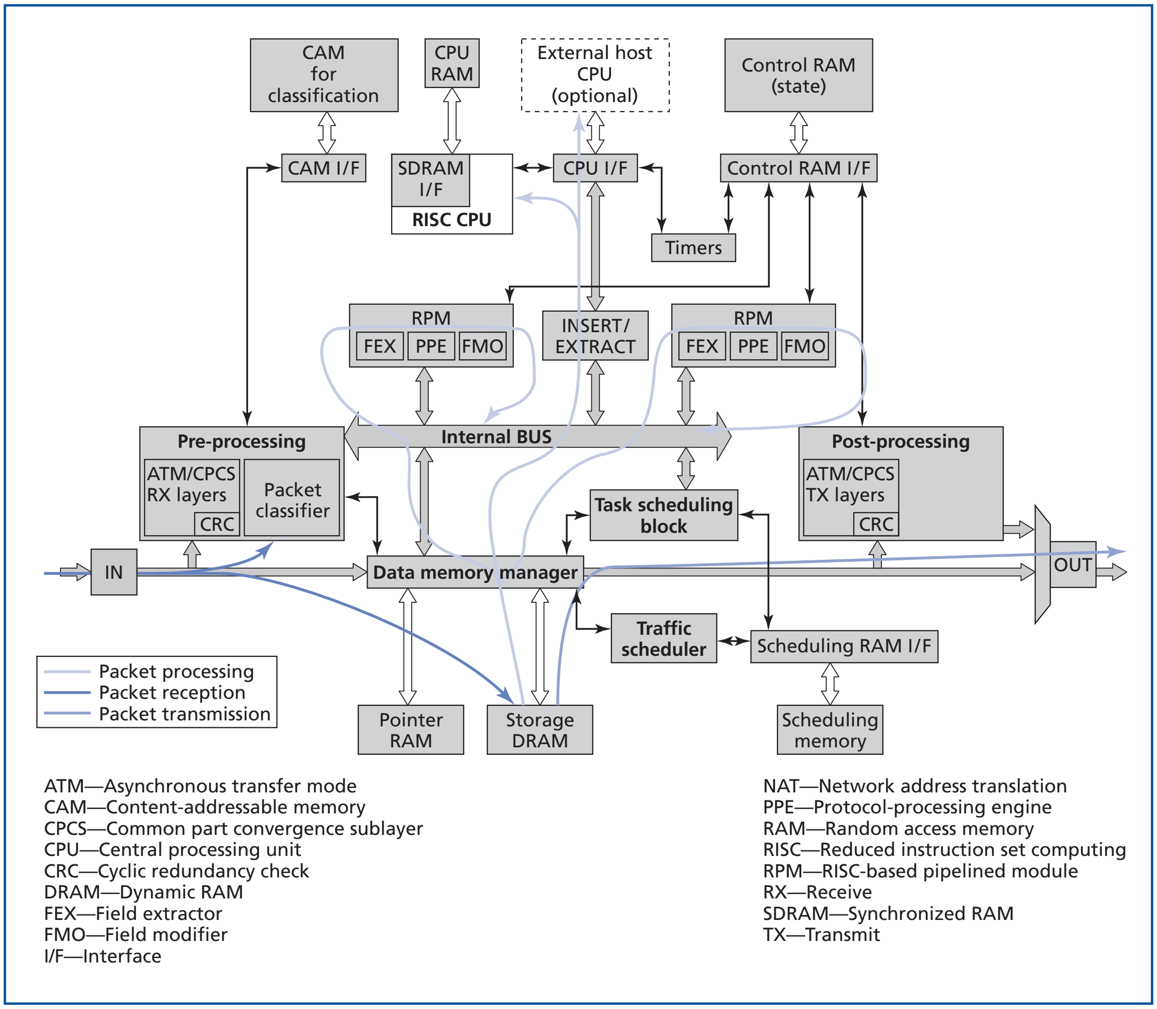


Figure 6.
Typical data flow in a stateful-inspection firewall system with NAT support.

architecture, which includes fixed hardware units and programmable engines designed to operate either as a pipeline or in parallel, we will define headroom as the percentage of the available processing resources of the chip that can be exploited in parallel. The main processing units that can operate in parallel are the two RPM units and the central RISC unit. Because the RPM (which receives the packet or packet header and executes the protocol message at wire speed) is the processing heart of the PRO3 architecture, its throughput—and that of its sub-block—determines application performance. The throughput of the RPM is determined by the worst-case performance of each of its pipeline stages, as we shall see in the following discussion.

We have evaluated several applications and have written the PRO3 specifications to meet the performance targets that are included in **Table II**. In applications 5 and 6 in Table II, maximum performance (i.e., 2.5 Gb/s sustained throughput) can be achieved for average-case conditions, based on typical IP packet distributions [21]. Worst-case conditions (i.e., a

Table II. PRO3 features.

	Application	Sustained rate	Max flows	Headroom (RISC, RPM1, RPM2)
	ATM applications			
1	ATM cell processing	2.5 Gb/s	512K	100%, 100%, 100%
2	AAL5 processing	2.5 Gb/s	512K	100%, 100%, 100%
	IP applications			
3	Layer 2, 3, 4 classification	2.5 Gb/s	512K	100%, 100%, 100%
4	Layer 2, 3, 4 filtering	≤2.5 Gb/s	512K	100%, 100%, 100%
5	Layer 4 stateful inspection	≤2.5 Gb/s	512K	100%, 0%, 0%
6	NAT	2.5 Gb/s	512K	100%, 0%, 0%

AAL5—ATM adaptation layer type 5
ATM—Asynchronous transfer mode
IP—Internet protocol
NAT—Network address translation
PRO3—Programmable protocol processor
RISC—Reduced instruction set computing
RPM—RISC-based pipelined module

continuous stream of minimum size 40-byte IP packets) have a negative impact on performance, as the analysis that follows in this section shows. However, this performance deterioration is to be expected, especially when a network processor must handle additional functions and even more complex applications.

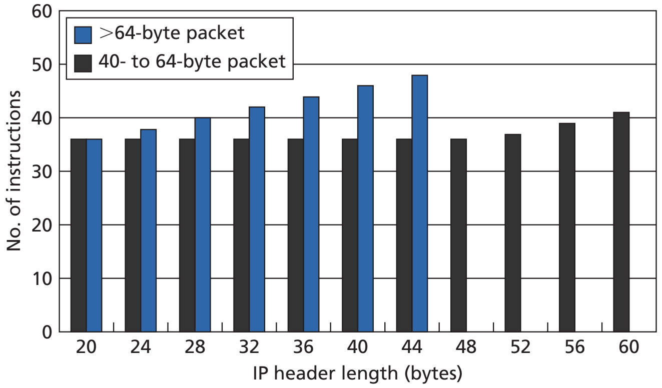
The performance evaluation of the stateful-inspection firewall with NAT support has been based on a PRO3 chip, implemented using UMC 0.18-micron CMOS technology, with a clock speed of 200 MHz and a 64-bit internal bus. Simulations were carried out on the final Verilog-flattened netlist, after the chip-layout procedure. Firmware was developed for all the microengines, and open-source C code was ported. Samples of real TCP/IP traffic were used as input, and the processing time of each of the programmable protocol engines of the RPM block was measured. In addition, in order to evaluate application performance, simulations were carried out with different packet and header lengths. Two parameters were measured: the total number of instructions executed and the corresponding processing time. Based on these figures, the throughput of each sub-block and of the whole module was estimated. Finally, the performance of each of the cores of the RPM module was investigated.

Simulation Results

The DMM sends the RPM either one 64-byte segment (if the IP packet length is no more than 64 bytes) or two (if the IP packet is larger than 64 bytes). In this manner, it is guaranteed that all the relevant fields from the IP and TCP headers will be sent for processing to the FEX programmable engine of the RPM module. **Figure 7** displays the total number of instructions executed by the FEX for varying IP header lengths and small (i.e., less than 64-byte) and large (i.e., greater than 64-byte) packets. For this particular implementation of the stateful-inspection firewall application, the FEX is supposed to process only the TCP and IP header of each packet, by extracting the fields listed in Table I.

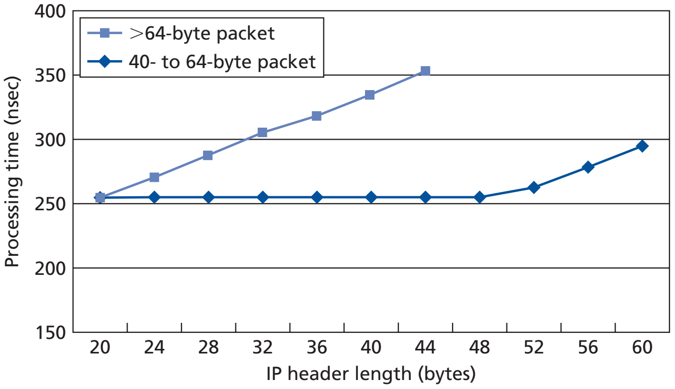
It is worth noting that, as shown in Figure 7, the number of instructions required is independent of the IP packet length. For small IP packets (i.e., those between 40 and 64 bytes), the number of instructions required is proportional to the IP header length (which is determined by the number of valid IP options). Also, for IP packets larger than 64 bytes, a fixed number of instructions is required when the IP header length is between 20 and 48 bytes, while for longer IP headers, the number of instructions increases proportionate to the IP header length, reaching a maximum of 43 instructions for an IP packet with 60 bytes of IP options. The average cycle-to-instruction ratio for the FEX microengine is 1.6. Although this value is not ideal, it can be improved by reducing the most clock-consuming instructions. However, since the processing does not depend on the total IP packet length, no great improvement in the value is possible.

Figure 7 also shows that the total number of instructions for two segments (i.e., IP packet length larger than 64 bytes) is smaller than the total number of instructions for one segment (i.e., IP packet length between 40 and 64 bytes). The reason for this is that the firmware easily identifies the case of two segments



FEX—Field extractor
IP—Internet protocol

Figure 7.
Number of FEX instructions executed for different IP header lengths.



FEX—Field extractor
IP—Internet protocol

Figure 8.
Processing time of FEX microengine for different IP header lengths.

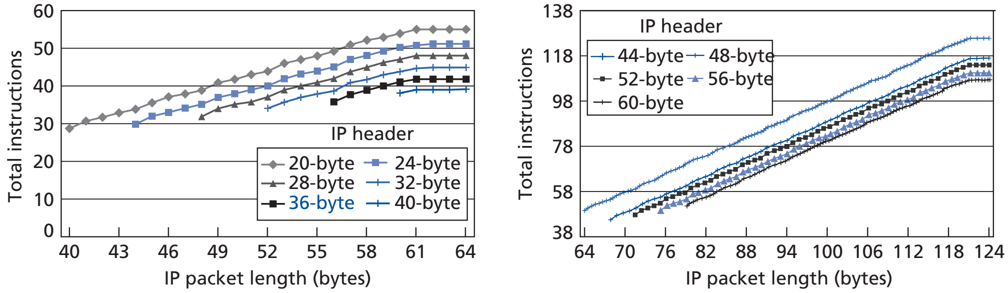
and, based on the IP header lengths (which appear in the first segment), scans faster and jumps directly to the fields to be extracted (which appear in the second segment). The relationship between the processing time of the FEX microengine and the IP header length is displayed in **Figure 8**. It can be seen that the throughput of the FEX microengine for 40-byte packets is close to 3.9 Mpackets per second (Mp/s). This throughput can be doubled when traffic is shared by the two RPM modules, allowing it to exceed the maximum 7.5 Mp/s throughput for 40-byte TCP/IP packets at the OC-48 line rate.

The FMO also receives the same number of segments as the FEX—one or two 64-byte segments—depending on the total packet size. However, in the FMO, the total processing time depends on the IP header length and the number of valid bytes in the segments (one or two stored in the bypass FIFO). To this end, optimization was possible, yielding significant improve in FMO sub-block performance. For example, the average cycle-to-instruction ratio was 2.2; after optimization it decreased to 1.7. This result was achieved by determining which firmware routines were called most often, which were the most clock-consuming routines, and which routines could be executed in parallel. After optimization, there was a significant decrease in the number of instructions executed, resulting in shorter processing times and in an improved cycle-to-instruction ratio.

Figure 9 displays the total number of instructions executed in the optimized FMO for varying total IP packet and IP header lengths. From Figure 9 it is apparent that, for packets having the same IP header length (i.e., the same number of valid IP options), the total number of FMO instructions required for NAT is, as expected, proportional to the IP packet length. Also, for IP packets that have the same length, the number of instructions is in inverse proportion to the IP header length (i.e., the number of valid IP options). The reason for this is that, the more IP options there are, the fewer jump instructions the microengine needs to scan the contents of the packet.

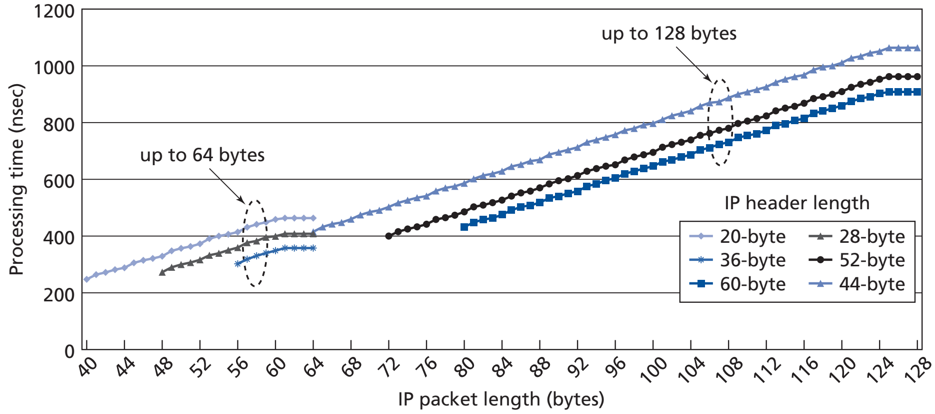
Finally, **Figure 10** shows how the total processing time varies with IP packet length. From this figure, it can be seen that a single FMO module can sustain about 4 Mp/s of traffic, assuming packets of 40 bytes. However, when the workload is balanced between the two RPM modules, the processing capability of both the FMO and the RPM exceeds the 7.5 Mp/s throughput required for processing continuous streams of minimum size (i.e., 40-byte) IP packets at the OC-48 link rate.

The performance of the third complex sub-block of the RPM module, the PPE, and, in particular, of its submodule, the modified Hyperstone RISC, depends heavily on the custom application that is running. It is estimated that, for each additional complex service in the firewall (e.g., TCP state updating), the modified RISC core needs around 170 instructions, which, of course, negatively affects overall system performance. However, for complex scenarios, this is a trade-off that any network processor faces. Our analysis indicates that, by using two RPM modules and balancing the load between them—which the design of the internal scheduler makes possible—it is possible to sustain 4 Mp/s in the worst case (i.e., only TCP traffic). For the average IP packet, which is about 128 bytes, this rate exceeds the OC-48 rate of 2.5 Gb/s. It is worth noting that packet classification, queuing, and scheduling can support 2.5Gb/s link rates in worst-case scenarios (i.e., with minimum packets). **Table III** gives the approximate number of instructions required for



FMO—Field modifier
IP—Internet protocol

Figure 9.
Number of FMO instructions executed for different IP packet lengths and different IP header lengths.



FMO—Field modifier
IP—Internet protocol

Figure 10.
FMO processing time for different IP packet lengths and different IP header lengths.

Table III. Number of MHY assembly instructions for TCP processing.

	No. of assembly instructions
Calculate and compare section	110
Switch section	60
Total	170

MHY—Modified Hyperstone RISC
RISC—Reduced instruction set computing
TCP—Transport control protocol

TCP processing, based on commercially available Hyperstone RISC processors and an analysis of the code developed for the implementation of the stateful-inspection firewall. The results in Table III are classified into two main sections: the first consists of a number of calculations and comparisons, and the second is a large switch statement section. It should be noted that not all the instructions of the switch section will be executed each time a TCP packet is processed, so the instruction count for the longest branch that might be executed would be lower than it might appear.

Conclusions

This paper has presented the PRO3 architecture, with emphasis on the implementation of a stateful-inspection firewall with NAT support. The PRO3 system uses the innovative concept of a three-stage pipelined module that integrates a RISC core with reconfigurable hardware on the same processing platform. In this way, a significant acceleration of protocol processing can be achieved in demanding applications like firewalls. Modern, stateful-inspection firewalls require increased processing power for handling a large number of concurrent connections at very high link rates, for monitoring all user-established flows, and for performing packet processing, filtering, and stateful inspection up to the application layer. PRO3 attempts to provide such a system with no degradation in network performance and, at the same time, to provide security adequate to prevent any attempts at illegal communication.

The PRO3 design is suitable for both cell- and packet-based network-processing applications with low memory requirements, and it has the flexibility to support multiple service disciplines in a programmable way, and to support thousands of flows. The PRO3 chip is being fabricated in UMC 0.18-micron CMOS technology occupying about 52 square millimeters; it will be packaged in a 1096 BGA package. Samples were delivered in August, 2002.

Acknowledgments

This work was performed within the framework of the PRO3 project, which is partially funded by the IST Program of the European Community. The author would like to acknowledge the contributions of his colleagues from Lucent Technologies, Hyperstone Electronics, IMEC, the National Technical University of Athens, Ellemedia Technologies, the Technical University of Crete, and InAccess Networks.

References

[1] Agere Inc., "Building Next Generation Network Processors," Mar. 2002, <http://www.interop.com/lv2002/ education/downloads/nps/np5_j_rolfe_w2.pdf>.

[2] Agere Inc., "The Challenge for Next Generation Network Processors," Mar. 2001, <http://www.agere.com/enterprise_metro_access/docs/challenge_new.pdf>.

[3] D. Buchanan, "True Wirespeed Firewalls Aid Security," Network World, 18:26 (2001), 49–53.

[4] W. Bux, W. Denzel, T. Engbersen, A. Herkersdorf, and R. Luijten. "Technologies and Building Blocks for Fast Packet Forwarding," IEEE Commun. Mag., 39:1 (2001), 70–77.

[5] P. R. Chandra and S. Y. Lim, "Framework for Benchmarking Network Processors," Draft 1.0, Network Processing Forum (Aug. 2002), <http://www.npforum.org/techinfo/NPFFramework.pdf>.

[6] C. A. Christiansen, N. S. Freedman, and C. Kolodgy, "Return of the Black Box: Firewall/VPN Security Appliances Unleashed," Internat. Data Corporation, Market Report, 2000.

[7] C. A. Christiansen and A. Germanow, "Worldwide Firewall Appliance Market: A Hot Market Creates New Competitors," Internat. Data Corporation, Market Report, 1999.

[8] C. Georgopoulos, G. Konstantoulakis, T. Orphanoudakis, N. Mouratidis, N. Zervos, N. Nikolaou, K. Pramataris, and J. Sanchez, "A Protocol Processing Architecture Backing TCP/IP-Based Security Applications in High Speed Networks," INTERWORKING 2000 (Bergen, Norway, 2000).

[9] L. Geppert, "The New Chips on the Block," IEEE Spectrum, 38:1 (2001), 66–68.

[10] Hyperstone Electronics E1-32XS RISC/DSP Processor, <http://www.hyperstone-electronics.com/downloads/pdf/E-32XSFlyer.pdf>.

[11] IP Filter, <http://coombs.anu.edu.au/~avalon/>.

[12] M. Katevenis, S. Sidiropoulos, and C. Courcoubetis, "Weighted Round Robin Cell Multiplexing in a General-Purpose ATM Switch Chip," IEEE J. on Selected Areas in Commun., 9:8 (1991), 1265–1279.

[13] R. Knobbe, A. Purtell, and S. Schwab, "Advanced Security Proxies: An Architecture and Implementation for High-Performance Network Firewalls," Military Commun. Conf. Proc., IEEE MILCOM 1999, 1 (1999), 734–738.

[14] S. Lavey, K. Funasaki, and A. M. Leibovitch, "Need for Next-Generation Switches and Routers Drives Network Processor Revolution," Internat. Data Corporation, Bulletin #19154 (May 1999).

[15] Linux IP-Chains, <http://www.tldb.org/HOWTO/IPCHAINS-HOWTO.html>.

[16] M. Nourani, G. Kavipurapu, and R. Gadiraju, "System Requirements for Super Terabit Routing," Circuits and Systems, 2001. MWSCAS 2001, Proc. 44th IEEE 2001, 2 (2001), pp. 926–929.

[17] N. Nikolaou, J. Sanchez, T. Orphanoudakis, D. Polatos, and N. Zervos, "Application Decomposition for High-Speed Network Processing Platforms," 2nd European Conf. on Universal Multiservice Networks, ECUMN 2002, (Colmar, France, 2002).

[18] A. Nikologiannis and M. Katevenis, "Efficient Per-Flow Queueing in DRAM at OC-192 Line Rate using Out-of-Order Execution Techniques," Proc. ICC2001 (Helsinki, Finland, 2001), pp. 2048–2052.

[19] P. Paulin, F. Karim, and P. Bromley, "Network Processors: A Perspective on Market Requirements, Processor Architectures and Embedded S/W Tools," Conf. on Design, Automation and Test in Europe, Munich, 2001.

[20] "The Programmable Protocol Processor, PRO3," <http://www.pro3-processor.com/>.

[21] K. Thompson, G. J. Miller, and R. Wilder, "Wide-Area Internet Traffic Patterns and Characteristics," IEEE Network, 11:6 (1997), 10–23.

[22] G. van Rooij, “Real Stateful TCP Packet Filtering in IP Filter,” Aug. 2001, <http://www.unixcircle.com/ipf/tcp_filtering.pdf>.

[23] K. Vlachos, T. Orphanoudakis, N. Nikolaou, S. Perissakis, D. Pnevmatikatos, G. Kornaros, J. A. Sanchez, and G. Konstantoulakis, “Processing and Scheduling Components in an Innovative Network Processor Architecture,” 16th Internat. Conf. on VLSI Design (New Delhi, India, 2003).

[24] J. Williams, “Architectures for Network Processing,” VLSI Technology, Systems, and Applications 2001, Proc. of Technical Papers, 61–64.

[25] R. Zalenski, “Firewall Technologies,” IEEE Potentials, 21:1 (2002), 24–29.

(Manuscript approved March 2003)

KYRIAKOS G. VLACHOS was formerly a member of technical staff in Bell Labs’ Advanced Technologies EMEA Department at Lucent Technologies in Hilversum, The Netherlands. He received his Dipl.-Ing. and Ph.D. degrees in electrical and computer engineering from the National Technical University of Athens (NTUA), Greece. He began his Bell Labs career in the Applied Photonics EMEA Department. Subsequently, he was a senior research associate in the Photonics Communications Research Laboratory. He has also been a consultant to the Hellenic Telecommunications Organization, assisting with the deployment of its new WDM backbone optical network. Dr. Vlachos has participated in various European R&D projects (e.g., DO-ALL, DAVID, STOLAS, PRO3, and HARMONICS) and in various national projects funded by the Greek government. His research interests are in the fields of all-optical digital logic, high-speed laser sources, optical transport networks, and optical packet switching. Dr. Vlachos is an author or co-author of 19 publications, and he is a member of the IEEE and the Technical Chamber of Greece. ◆