CHAPTER 34

NETWORK PACKET PROCESSING IN RECONFIGURABLE HARDWARE

John W. Lockwood
Washington University in St. Louis and Stanford University

This chapter will show, through an example, how networking systems have been built with reconfigurable hardware. It will describe how data can be switched, routed, buffered, processed, scanned, and filtered over networks using field-programmable gate arrays (FPGAs).

The chapter begins by describing the mechanisms by which Internet packets are segmented into frames and cells for transmission across a network. Internet Protocol (IP) wrappers are introduced, and it is shown how they simplify the implementation of large packet-processing systems. Next, a framework for building modular systems that implement Internet firewalls and intrusion prevention systems is presented. The chapter continues with a detailed explanation of how Bloom filters can scan streams of data for fixed strings and how finite automata can be used to scan for regular expressions.

Case studies are provided that show how deep packet inspection systems are implemented in reconfigurable hardware. One circuit detects the spread of worms and viruses across an Internet link. Another circuit analyzes the semantics of the text in traffic flows to determine which language is used within attached documents. A hardware-accelerated version of the popular SNORT intrusion detection system is illustrated, and it is shown how the FPGA hardware works with the software on a host to analyze packets.

34.1 NETWORKING WITH RECONFIGURABLE HARDWARE

34.1.1 The Motivation for Building Networks with Reconfigurable Hardware

Although modern microprocessors continue to improve their performance, they are not improving as fast as the rate at which data flows over Internet connections. As the limits of Moore's Law are reached, alternative computational methods are needed to route, process, filter, and transform Internet datastreams.

Networking systems created with reconfigurable hardware are flexible and easily modified to provide new functionality. Reconfigurable hardware enables features on networking platforms to be implemented in ways that are quite different from current platform implementations. It allows new modular components to be created and then dynamically installed in remote networksystems.

By processing network packets in hardware rather than in software, networking applications do not suffer the performance penalty caused by sequential data processing.

The Internet evolves as new protocols, features, and capabilities are added to the routers that implement the underlying network. Protocols, such as IP version 6 (IPv6), allow more devices to be individually addressed. Added features, such as per-flow queuing, allow voice and video to be reliably delivered in real time. Firewalls and intrusion prevention systems (IPSs) enhance Internet security.

Network platforms have been built to route network traffic, filter packets, and queue data in reprogrammable hardware. With reconfigurable hardware, networking platform operation can change over time as packet-processing algorithms and protocols evolve. With FPGAs, all features of the packet-processing system are configurable down to the logic gates. These systems enable new services to deploy and operate at the rate of the highest-speed backbone links.

34.1.2 Hardware and Software for Packet Processing

For their packet-processing operations, today's fastest routers use network processing elements implemented in custom silicon or in application-specific integrated circuits (ASICs). As shown in Figure 34.1, network processing elements reside between the line card where packets are transmitted and received and the Gigabit/second rate switch fabric that interconnects ports. They contain hundreds to thousands of parallel logic circuits and finite-state machines that are optimized to route, filter, queue, and/or process Internet datagrams in hardware.

Several platform types have been developed, many of which use standard microprocessors such as the Intel Pentium, AMD Athlon, or Motorola/IBM PowerPC. Others use ASICs from vendors such as Agere, Intel, Motorola, Cavium, Broadcom and Vitesse. Although software-based systems have outstanding flexibility, their packet processing is limited because of the sequential nature of their instruction execution. ASICs and custom silicon networking chips have high performance, but they offer little flexibility as measured by their ability to reprogram. Figure 34.2 illustrates the trade-offs between flexibility and performance.

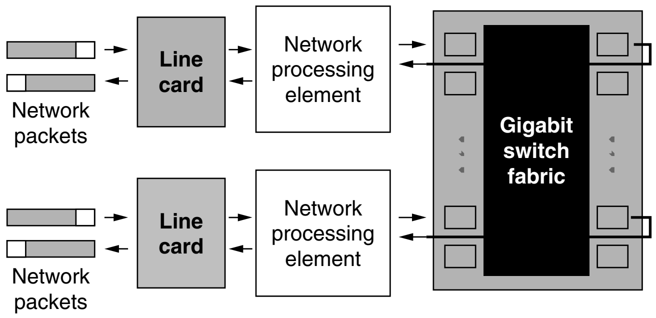

FIGURE 34.1 ■ A reconfigurable network processing element located between a line card and switch fabric.

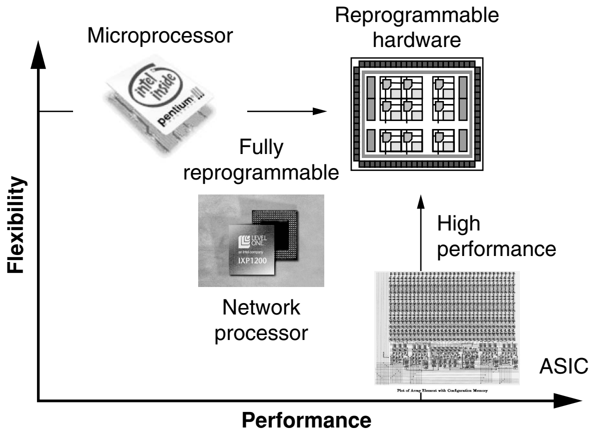

FIGURE 34.2 ■ Flexibility and performance trade-offs for networking systems that use microprocessors, network processors, ASICs, and reprogrammable hardware.

34.1.3 Network Data Processing with FPGAs

Reconfigurable hardware devices share the performance advantage of ASICs because they can implement parallel logic functions in hardware. However, they also share the flexibility of microprocessors and network processors because they can be dynamically reconfigured.

Using FPGAs for high-performance asynchronous transfer mode (ATM) networking was explored during the development of the Illinois Pular-based Optical Interconnect (iPOINT) testbed. In this project, an ATM switch with FPGAs [2] was developed and an advanced queuing module was implemented that provided per-flow queuing functionality in FPGA hardware. The FPGAs were used to implement the datapath of the switch and to control the state machines that buffered the ATM cells as they arrived on each switch port of the switch. The lookup tables (LUTs) in the FPGA fabric were used to build the multiplexers that switched the data between the ports. Finally, combinational logic was used to implement the state machines that controlled how packets were written to and read from SRAM [3].

FPGAs have also proven effective for implementation of bit-intensive function networking, such as forward error correction (FEC), and for boosting the performance of networking protocols [4]. The bitwise processing function maps well into the fine-grained logic on an FPGA. On-chip LUTs are used to encode data patterns as symbols with redundant bits of information. When the symbols are decoded, the redundant bits allow the receiver to reconstruct the data even with a few bits in error. Reconfigurable logic allows algorithms that use varying amounts and types of error correction to be programmed on-chip.

Through the development of the Field-Programmable Port Extender (FPX) platform [1], it was demonstrated that high-performance network packet-processing systems implemented with FPGAs are both useful and practical. The

FPX platform used two multi-Gigabit/second network interfaces, four banks of off-chip memory, and two FPGAs to implement over 30 networking applications. Applications developed for the FPX platform included modules that performed Internet Protocol IP address lookup for routing [7]; payload scanning for detection of fixed strings and regular expressions within the body of a packet; data queuing to provide quality of service (QoS); intrusion detection to determine when a network may be under attack; intrusion prevention to halt such attacks; and semantic processing of network data.

34.1.4 Network Processing System Modularity

Modularity is a key feature of networking systems. Network developers need standard interfaces to interface high-level network processing components to the underlying network infrastructure. In systems with reconfigurable hardware, modules can be implemented in regions of an FPGA and bound by a well-defined interface to the datapath and to external memory. Multiple modular data-processing components can be integrated to compose systems. Memory interfaces can connect logic to off-chip memory in order to buffer data and hold large lookup tables LUTs.

For the FPX platform modules, data was received and transmitted via a series of ATM cells carried over a 32-bit-wide Utopia interface. ATM cells contained 48 bytes of payload data and 4 bytes of a header that included a virtual path identifier (VPI) and a virtual circuit identifier (VCI). Each ATM cell also included an 8-bit checksum that covered the ATM cell header. Larger IP datagrams were sent between modules using layered protocol wrappers that segmented and reassembled multiple cells into ATM adaptation layer 5 (AAL5) frames. These frames contained data from a series of ATM cells and a 32-bit checksum at the end that covered all bytes of the payload. Segmentation and reassembly of cells into frames were performed to transfer packets over the network.

The FPX platform (Figure 34.3) stored and loaded data from two types of off-chip memory. Two interfaces supported transfer of 36-bit-wide data to and from an on-chip SRAM. SDRAM interfaces provided 64-bit-wide interfaces to multiple banks of high-capacity, off-chip memory. In the implementation of the IP lookup module, the off-chip SRAM was used to store data structures for IP lookup, while the SDRAM was used to buffer packets. The lower latency of SRAM access was important for the implementation of lookup functions where there was a data dependency for the result; the larger capacity of the SDRAM was beneficial for reducing the cost of storing bulk data, including buffering dataflows.

A switch was implemented using the reprogrammable application device (RAD) FPGA logic that allowed traffic to be routed to extensible modules. Layered protocol wrappers performed the segmentation and reassembly of AAL5 frames so that full packets could be processed by the FPGA hardware. To reprogram the RAD FPGA that contained the extensible modules, configuration and control logic was implemented on the network interface device (NID) FPGA.

The FPX platform was integrated into the Washington University Gigabit Switch (WUGS) to process packets as they passed into and out of the networking

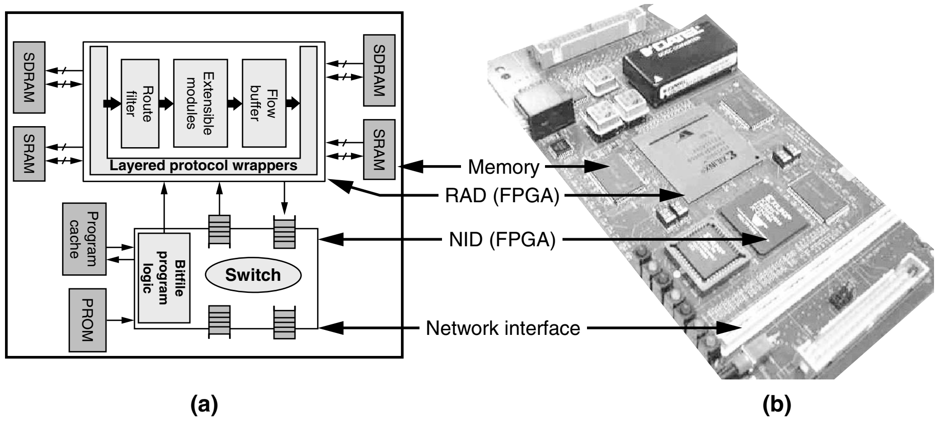

FIGURE 34.3 ■ A block diagram and a physical implementation of the FPX platform.

ports of a scalable network switch. The WUGS switching platform provided a backplane for transferring ATM cells between ports. By adding the FPX between the line cards and the switch fabric, the system was able to analyze, process, route, and filter IP packets as they flowed through the system. OC-3 to OC-48 line cards were used to directly send and receive ATM cells, while Gigabit Ethernet line cards were used to segment frames into multiple ATM cells and reassemble them. After data passed through the FPX, they were forwarded to the switch fabric, where cells were forwarded to other FPX modules in the chassis based on their VPI and VCI values.

34.2 NETWORK PROTOCOL PROCESSING

The Open Systems Interconnection (OSI) Reference Model defines how multiple layers can be used to transport data over a computer network. OSI divides the functions of a protocol into a series of layers, each of which has two properties: (1) It uses only the functions of the layer below, and (2) it exports functionality only to the layer above. A system that implements protocol behavior consisting of a series of these layers is known as a protocol stack. Protocol stacks can be implemented in hardware, in software, or in a mixture of the two (typically, only the lower layers are implemented in hardware; the higher layers, in software). This logical separation makes reasoning about the behavior of protocol stacks much easier and allows their design to be elaborate but highly reliable. Each layer performs services for the next highest layer and makes requests for the next lowest layer [5].

For real systems that process Internet data, the OSI model is not directly implemented but instead serves as a reference for implementation of the real protocols. Layers are important for processing IP data, however, because they permit application-processing modules to abstract details of the lower-layer

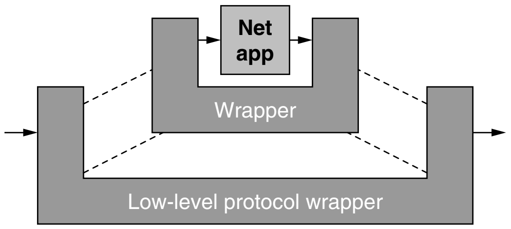

FIGURE 34.4 ■ Integration of a network application within one or more wrappers.

network protocols. At the lowest layer, networks modify raw cells of data that move between interfaces. At higher layers, the applications process variable-length frames or IP packets. To send and receive data at the user level, a network application may transmit directly or receive user datagram protocol (UDP) messages by instantiating all wrappers and sending data from a network application down through a series of wrappers [6] (see Figure 34.4).

34.2.1 Internet Protocol Wrappers

Hundreds of millions of computers deployed throughout the world communicate over the Internet. Traffic from these machines is concentrated to flow over a smaller number of routers that forward traffic through the Internet core. Currently, Internet backbones operate over communication links ranging in speed from OC-3 (155 Mbps) to OC-768 (40 Gbps). Fast links that process small packets have the ability to process millions of IP packets per second.

A library of layered protocol wrappers (see Figure 34.5) was developed to process Internet packets in reconfigurable hardware. Collectively, the wrappers simplified and streamlined the implementation of high-level networking functions by abstracting the operation of lower-level packet-processing functions. The library infrastructure was synthesized into FPGA logic and integrated into an FPX network platform. At the lowest levels, the library processes ATM cells. Complete frames of data are segmented and reassembled using ATM adaptation layer 5 (AAL5), over which IP messages are then transported.

When only a single message needs to be transmitted, the UDP can send one packet over the Internet. UDP encapsulates a variable-length message into an IP packet and allows the system to specify source and destination port numbers that identify from which application on a machine the data was sent and to which application it should be delivered. UDP/IP also provides a checksum to ensure the integrity of the data. Using the FPX protocol-processing library, this checksum is automatically computed, using FPGA hardware, as the sum over the payload bytes of the message.

34.2.2 TCP Wrappers

Over 85 percent of all traffic on the Internet today uses the Transmission Control Protocol (TCP). TCP is stream oriented and guarantees delivery of data with

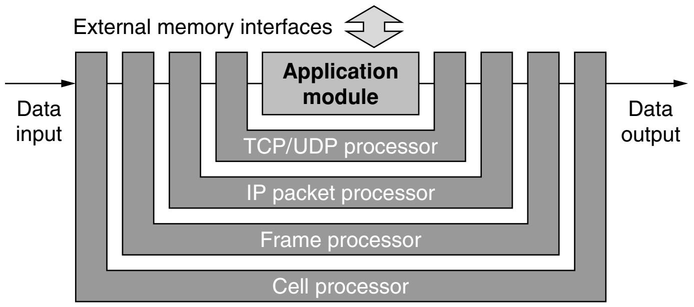

FIGURE 34.5 ■ Implementation of layered protocol wrappers on the FPX platform.

an ordered byte flow. Processing TCP dataflows in the middle of the network is extremely difficult because network packets can be dropped, duplicated, and reordered. Packet sequences observed within the interior of the network may be different from packets received and processed at the connection endpoints. The complexities associated with tracking the state of end systems and reconstructing byte sequences based on observed traffic are significant.

A TCP processing circuit was developed that handles the complexities associated with flow classification and TCP stream reassembly. It provided the FPGA logic with a view of network traffic flow data through a simple client interface. The TCP wrapper enabled other high-performance data-processing subsystems to operate on TCP network content without needing to implement their own state-tracking operations. The TCP module used a state store to track the status of each TCP/IP flow and, using a hash function, assigned a unique flow number to each session [8].

Figure 34.6 is a block diagram of the TCP processor. Internet packets arrive as frames of data to the input state machine of the TCP processing engine. The input state machine forwards the frames to a first in, first out (FIFO) that buffers the packet; a checksum engine that computes and verifies the correctness of the TCP checksum; and a flow classifier that computes a flow identifier (flow ID) using a hash over fields in the packet header.

The flow ID is passed to the state store manager that retrieves the state associated with the particular flow. Results are written to the control and state FIFO, and the state store is updated with the current flow state. The output state machine reads data from the frame and control FIFO buffers and passes data to the packet-routing engine. Most traffic flows through the content-scanning engines, which scan the data. Packet retransmissions bypass these engines and go directly to the flow-blocking module.

Data returning from the content-scanning engines also goes to the flow-blocking module. This stage updates the per-flow state store with application-specific state information. If a content-scanning engine indicates that it has a need to block a flow, the flow-blocking module can enforce this rule by comparing the packet's sequence number with the sequence numbers for which flow blocking should take place. If the packet meets the blocking criteria, the

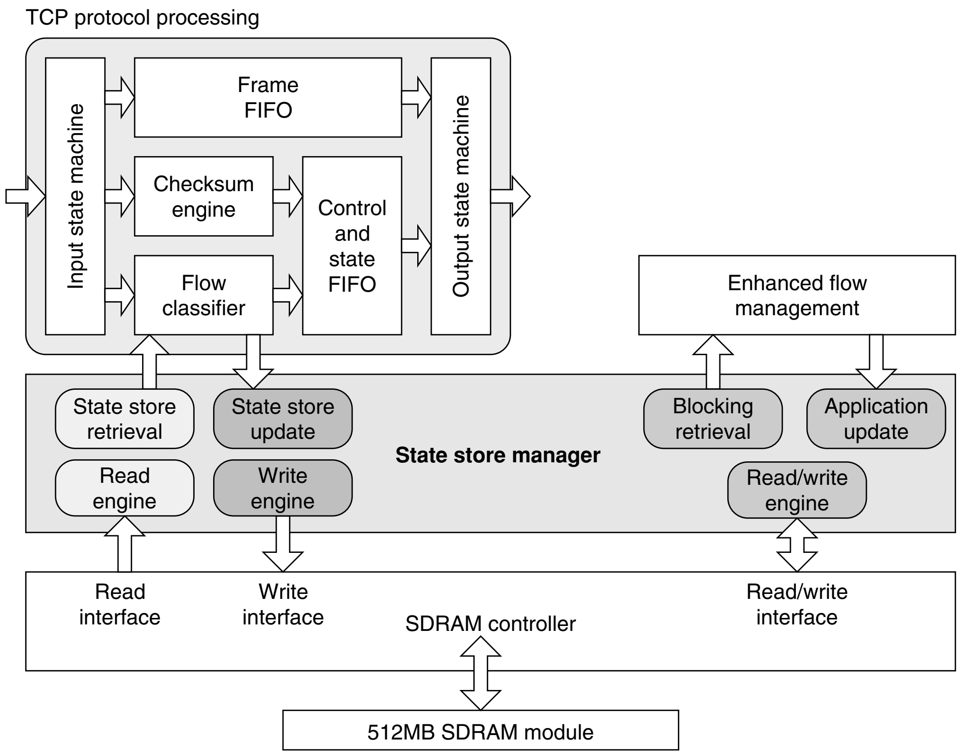

FIGURE 34.6 ■ A block diagram of the TCP processor.

flow-blocking module drops it from the network. Any remaining packets go to the outbound protocol wrapper.

The state store manager processes requests to read and write flow state records. It also handles all interactions with SDRAM memory and caches recently accessed flow state information. The SDRAM controller exposes three memory access interfaces: a read/write, a write-only, and a read-only. The controller prioritizes requests in that order, with the read/write interface having the highest priority.

34.2.3 Payload-processing Modules

Many network applications have a common requirement for string matching in the payload of packets or flows. Once the data being transported over the network has been reconstructed using the IP and TCP modules, it can be examined in the payload. For example, the presence of a string of bytes (or a signature) can identify the presence of a media file, an attachment, or a security exploit. Well-known Internet worms, such as Nimda, Code Red, and Slammer, propagate by sending malicious executable programs identifiable by certain byte sequences in payloads [14]. Because the location (or offset) of such strings and their length are unknown, such applications must be able to detect strings of different lengths starting at arbitrary packet payload locations.

Packet inspection applications, when deployed at router ports, must operate at wire speeds. As network rates increase, the implementation of packet monitors that process data at Gigabit/second line rates has become increasingly difficult. Thus, the growth in network traffic has motivated specialized packet- and payload-processing modules in hardware.

34.2.4 Payload Processing with Regular Expression Scanning

A regular expression (RE) is a pattern that describes a set of strings. The basic building blocks for these patterns consist of individual characters, such as {`a`, `b`, and `c`}. These characters can be combined with meta-characters, such as: {`*`, `|`, and `?`}, to form regular expressions with wildcards. For two regular expressions, `r1` and `r2`, rules define that `r1*` matches any string composed of zero or more occurrences of `r1`; `r1?` matches any string composed of zero or one occurrence of `r1`; `r1|r2` matches any string composed of `r1` or `r2`; and `r1r2` matches any string composed of `r1` concatenated with `r2`. For instance, `a` is an RE that denotes the singleton set {`a`}, while `a|b` denotes the set {`a`, `b`} and `a*` denotes the infinite set {null, `a`, `aa`, `aaa`, . . .}. REs can be identified using nondeterministic finite automata (NFA).

Research on RE matching in hardware has been performed by Sidhu and Prasanna [16] and Franklin et al. [17]. Sidhu and Prasanna were primarily concerned with minimizing the time and space required to construct NFAs. They ran their NFA construction algorithm in hardware as opposed to software. Franklin et al. followed with an analysis of this approach for the large set of expressions found in a SNORT database [18].

The search function FPgrep was implemented by Moscola et al. to search packet payloads for substrings that belong to the language defined by the RE [15]. When FPgrep matched a substring in a packet, it transmitted information about the packet to a monitoring host system. The information sent for network intrusion detection functions specified the content found and the sender's and receiver's IP addresses. The search ran in linear time (proportional to packet size), $O(n)$ (where n was the number of bytes in a packet), and in constant space. That is, there was never a need to examine a character more than once and the amount of hardware was proportional to the size of the RE. Approximately one flip-flop was required per character.

A streaming content editor, FPsed, was implemented as a module on the FPX platform. The FPsed module selectively replaced content in packet payloads. String replacement for an RE is not as straightforward or efficient as searching. It requires that the machine do more than simply determine the presence of matching substrings in a record—it must also determine the position of the first and last character of all complete substrings that are matched by it. It is this requirement that makes RE search and replace more complicated and less efficient than a simple search. Searching for the complete substring is logical when the goal is to replace it.

Consider the replacement of every occurrence of a certain hexadecimal string associated with a computer virus, `3n*4n*5n*B`, with the text `Virus Pattern Detected`. For the sake of brevity, the previous expression uses *n* as shorthand for any hexadecimal character (i.e., `0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F`). For the input string `3172F34435B6B7B8`, the substring can be replaced from the point where the machine starts running, `34`, to the point where the substring is accepted, just before `B6` (i.e., substring `34435B`). However, this would allow a portion of the virus to remain in the content stream. In most situations, it is preferable to replace complete substrings; here the complete substring match starts with `31` and includes everything to just before `B8` (i.e., the substring `3172F34435B6B7B`).

34.2.5 Payload Scanning with Bloom Filters

A hash table is one of the most attractive choices for quick lookups. Hash tables require only constant time, $O(1)$, average memory accesses per lookup. Because of their versatile applicability in network packet processing, it is useful to implement these hashing functions in hardware [19, 20].

Bloom filters can detect strings of characters that appear in streaming data moving at very high data rates. A Bloom filter is a data structure that stores a set of signatures compactly by computing multiple hash functions on each member of the set. It queries a database of strings to check for the membership of a particular string. The answer to this query can be false positive but never false negative. The average computation time to perform a query remains constant so long as the sizes of the hash tables scale linearly with the number of strings they store. Because each table entry stores only a hashed version of the content, the amount of storage required by the Bloom filter for each string is independent of its length.

34.3 INTRUSION DETECTION AND PREVENTION

Existing firewalls that examine only the packet headers do little to protect against many types of attack. Multiple new worms transport their malicious software, or *malware*, over trusted services and cannot be detected without examining the payload. Intrusion detection systems (IDSs) perform deep scanning of the payload to detect malware, but do nothing to impede the attack because they only operate passively. An intrusion prevention system (IPS), on the other hand, can intervene and stop malware from spreading. The configuration of a network intrusion prevention system is shown in Figure 34.7.

One problem with software-based IDSs is that they cannot keep pace with the high volume of traffic that transits high-speed networks. Existing systems that implement IPS functions in software limit the bandwidth of the network and delay the end-to-end connection.

A reconfigurable system that can keep pace with high-speed network traffic has been developed. It scans data quickly, reconfigures to search for new attack

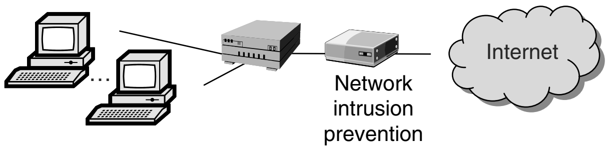

FIGURE 34.7 ■ Configuration of an in-line network IPS situated between two hosts attached to a router and to the Internet.

patterns, and takes immediate action when attacks occur. By processing the content of Internet traffic in real time within an extensible network, data that contains computer viruses or Internet worms can be detected and prevented. By adding only a few filtering devices at key network aggregation points, Internet worms and computer viruses can be quarantined to the subnets where they were introduced.

A complete system has been designed and implemented that scans the full payload of packets to route, block, and track the packets in the flow based on their content. The result is an intelligent gateway that provides Internet worm and virus protection in both local and wide area networks.

Network intrusion detection and prevention systems search for predefined virus or worm signatures in network traffic flows (see Section 34.2.3). Such signatures can be loaded into the system manually by an operator or automatically by a signature detection system. (Note that *string* is synonymous with *signature* throughout the chapter.)

Once a signature is found, an intrusion detection and prevention system (IDPS) can use it to block traffic containing infected data from spreading throughout a network. To perform this operation on a high-speed network, the signature scanning and data blocking must operate quickly. Comparing a variety of systems running the SNORT rule-based NID sensor reveals that most general-purpose computer systems are inadequate as NID sensor platforms even for moderate-speed networks. Factors such as microprocessor, operating system, main memory bandwidth, and latency limit the performance that an NIDS sensor platform can achieve [22].

34.3.1 Worm and Virus Protection

Computer virus and Internet worm attacks are pervasive, aggravating, and expensive, both in terms of lost productivity and consumption of network bandwidth. Attacks by Nimba, Code Red, Slammer, SoBig.F, and MSBlast have infected computers globally, clogged large computer networks, and degraded corporate productivity. It can take weeks to months for information technology professionals to sanitize infected computers in a network after an outbreak [24].

In the same way that a human virus spreads among people coming in contact with each other, computer viruses and Internet worms spread when computers communicate electronically [25]. Once a few systems are compromised, they infect other machines, which in turn quickly spread the infection throughout a network. As is the case with the spread of a contagious disease, the number of infected computers grows exponentially unless contained. Computer systems spread contagion much more quickly than humans do because they can communicate instantaneously over large geographical distances. The Blaster worm, for example, infected over 400,000 computers in less than five days. In fact, about one in three Internet users are infected with some type of virus or worm every year.

Malware can propagate as a computer virus, an Internet worm, or a hybrid of both. Viruses spread when a computer user downloads unsafe software, opens a malicious attachment, or exchanges infected computer programs over a network. An Internet worm spreads over the network automatically when malware exploits one or more vulnerabilities in an operating system, a web server, a database application, or an email exchange system.

Malware can appear as a virus embedded in software that a user has downloaded. It can also take the form of a Trojan that is embedded in what appears to be benign freeware. Alternatively, it can spread as content attached to an email message, as content downloadable from a web site, or in files transferred over peer-to-peer systems. Modern attacks typically use multiple mechanisms to execute. Malware, for example, can spoof messages that lure users to submit personal financial information to cloaked servers. In the future, malware is likely to spread much faster and cause much more damage.

Today, most anti-virus solutions run in software on end systems. To ensure that an entire network is secure from known attacks, integrated systems were developed that can perform multiple network processing functions.

34.3.2 An Integrated Header, Payload, and Queuing System

An integrated system that incorporated the payload-scanning function, a ternary content addressable memory (TCAM) for header matching, and a flow buffer and queue manager for packet storage was implemented [13]. It is shown as a block diagram in Figure 34.8.

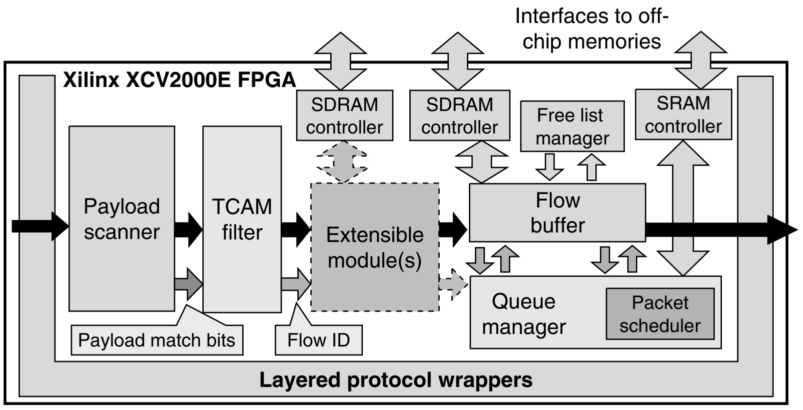

FIGURE 34.8 ■ Complete on-chip networking header and payload processing integrated with a flow buffer and a queue manager.

SNORT is a lightweight NID sensor that can filter packets based on predefined rules over packet headers and payloads [18]. With the TCP option enabled, SNORT matches strings that appear anywhere within traffic flows. Each SNORT rule operates first on the packet header to verify that the packet is from a source or to a destination network address and/or port of interest. If the packet matches a certain header rule, its payload is scanned against a set of predefined patterns associated with that rule. Matching of one or multiple patterns implies a complete match of a rule, and further action can be taken on either the packet or the TCP flow.

To provide complete detection of all known attacks, an intrusion system must process all packets. Several thousand patterns appeared in the version 2.2 rule set for SNORT. SNORT's rule database continually expands as new threats are observed. As the number of headers and signatures to match increases, the CPU on a PC running SNORT becomes overloaded and not all packets are processed.

A SNORT intrusion filter for TCP (SIFT) was implemented in reconfigurable hardware and is illustrated in Figure 34.9. SIFT data entered the system via the TCP de-serialize wrapper. Control signals marked specific locations in the

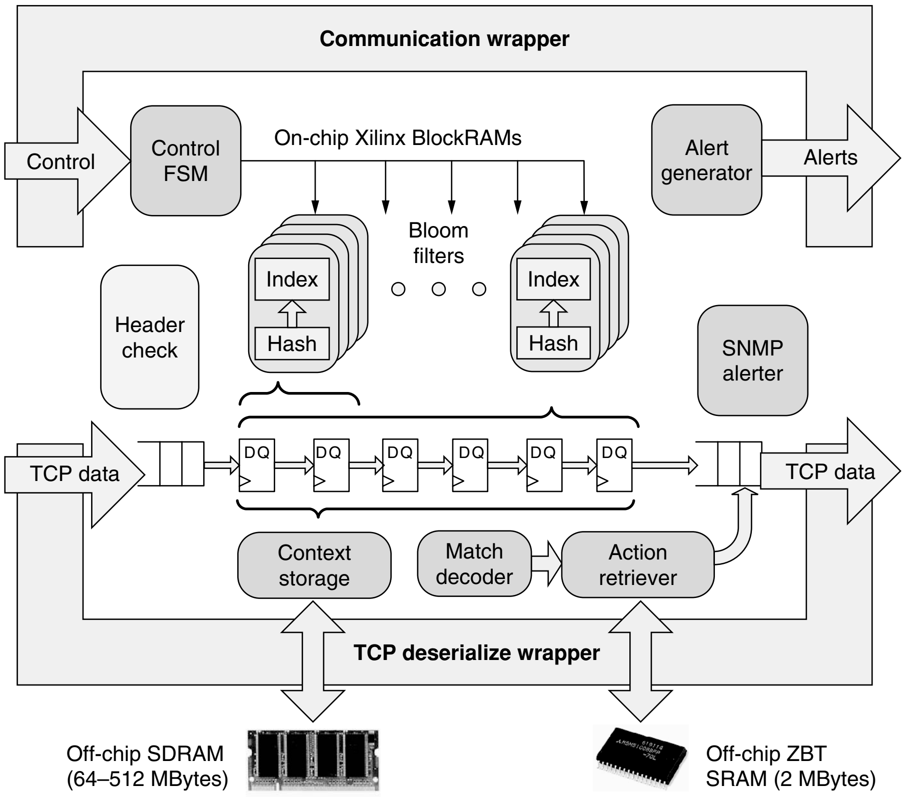

FIGURE 34.9 ■ A block diagram of SIFT.

packet that included the starts of the IP header, the TCP header, and the payload. The value of the header was sent to a header check component to determine if the packet matches a header-only rule. The payload was sent through an 8-stage pipeline where each byte offset is searched for signatures by Bloom filters. If a match was detected, the match decoder determines the string identifier (ID), which was next sent to the action retriever to determine what to do with the packet. Suspect packets were forwarded to software for further inspection. Those that had no match were not inspected further; those that did need additional processing were sent to the outgoing side of the TCP de-serialized wrapper.

To match payloads, SIFT used Bloom filters to allow signatures to be incrementally programmed into hardware. Signatures could be added or deleted via messages embedded in UDP control packets. These packets were sent through the communication wrapper to a control finite-state machine (FSM). In turn, the FSM set the appropriate bits in BlockRAM memories on the FPGA to add the signature to the Bloom filter. To achieve high throughput, four engines ran in parallel [21].

34.3.3 Automated Worm Detection

Outbreaks of new worms constitute a major threat to Internet security. IDPSs described previously only filter traffic that contain known worms. Systems that automatically detect new worms in real time by monitoring traffic on a network allow detection and protection from new outbreaks.

Internet worms spread by exploiting vulnerabilities in operating systems and application software that run on end systems. Once they infect a machine, they use it to attack other hosts; these attacks compromise security and degrade network performance, causing large economic losses for businesses resulting from system downtime and lowered worker productivity. The Susceptible/Infective (SI) model illustrates the spread of Internet worms [25]. With this model, a well-known equation can be used to estimate how fast a worm will infect vulnerable machines.

Worms can be prevented by writing code that has no vulnerabilities, and the computer security community has made great strides toward this goal. Programmers analyze the vulnerability that the worm exploits and release a "patch" to fix it. However, it takes time to analyze and patch software. In addition, many end users may never apply the patch, and as a result a significant number of machines in the network remain vulnerable.

Another way to prevent the spread of worms is to have the network contain them. When intrusion prevention systems scan traffic for a predetermined signature and filter the flows that match, the spread of a known worm can be blocked. The EarlyBird System [26, 27] detects the signatures for unknown worms in real time, identifying them by their repeating content. Because worms consist of malicious code, frequently repeated content on the network can be a useful warning of worm activity. Large flows are identified by computing a hash of packet content in combination with a destination port.

A hardware-accelerated worm detection circuit implemented in reconfigurable hardware draws from two ideas presented in the EarlyBird system [23]. To detect commonly occurring content, a hash is computed over 10-byte windows of streaming data. The hash value is used to identify a counter in a vector that is instructed to increment by one. At periodic intervals (called timeouts), the counts in each of the vectors are decremented by the average number of arrivals due to normal traffic. When a counter reaches a predetermined threshold, an alert is generated and its value is reset to zero.

For the implementation of the circuit on an FPGA, the count vector was implemented by configuring dual-ported, on-chip BlockRAMs as an array of memory locations. Each memory afforded one read operation and one write operation every clock cycle, which allowed a 3-stage pipeline to be implemented that reads, increments, and writes memory every clock cycle. Because the signature changes every clock cycle and because every occurrence of every signature must be counted, the dual-ported memories allow the occurrence count to be written back while another count is being read.

When an on-chip counter crosses the threshold, the corresponding signature is hashed to a table in off-chip SRAM. The next time the same string causes the counter to exceed the threshold, it is hashed to the same location in SRAM and the two strings are compared. If they are the same, it is determined that the match is not a false positive and the counter is incremented. If they are different, the contents of the string stored in SRAM is overwritten with the value of the new string and the count is reset.

On receiving confirmation from the SRAM analyzer that a signature frequently occurs, a UDP control packet is sent to an external computer. The packet contains the offending signature, which is the string of bytes by which the hash was computed. The computer, in turn, programs other IDS/IDP systems to filter traffic that contains this signature.

34.4 SEMANTIC PROCESSING

Next-generation networks route and forward data based on the semantics of the data within documents. Rather than assigning arbitrary headers to packets, routers use the meaning of the text itself to determine the packet routing.

34.4.1 Language Identification

As of 2004, nearly two-thirds of the world's Internet users spoke a non-English native language [29], and nearly one-third of the pages available on the Internet were written in a non-English language [29, 30]. As the rate at which data is transferred over the Internet increases, the rapid identification of languages becomes an increasingly difficult problem. A system capable of quickly identifying the primary language or languages used in documents can be useful as a preprocessor for document classification and translation services. It can also be used as a mechanism for language-based document routing.

A hardware-accelerated algorithm was designed to automatically identify the primary languages used in documents transferred over the Internet [28]. The module was implemented in hardware on the FPX platform. Referred to as Hardware-Accelerated Identification of Languages (HAIL), this complete system identified the primary languages used in content transferred over TCP/IP networks. It operated on streaming data at a rate of 2.4 Gigabits/second using FPGA hardware. This level of performance far outstripped software algorithms running on microprocessors.

Several methods have been shown to be effective for the classification of document characteristics based on principles from linguistics and artificial intelligence. Some methods used dictionary-building techniques [31], while others used Markov Models, trigram frequency vectors [32], and/or *n*-gram–based text categorization [33, 34]. Although these methods are capable of achieving high degrees of accuracy, most require floating-point mathematics, large amounts of memory, and/or generous amounts of processing time.

HAIL uses *n*-grams to determine the language of a document. These are sequential patterns of exactly *n* characters that are found in written documents, and when they are used as indicators of language, the primary language or languages of a document can be reliably determined. HAIL can use any *n*-gram length, although experiments have shown that *n*-grams of length 3 (trigrams) and length 4 (tetragrams) provide the most accurate results.

Before processing data with HAIL, the target system is trained with information on languages. Training is performed by scanning a set of documents in the languages of interest. When an *n*-gram appears significantly more frequently in the documents of one language than in any other, it is associated with that language. After training has established which *n*-grams best correspond to particular languages, memory modules on the hardware platform implementing HAIL have to be programmed. Memory is populated by using a hash to map each *n*-gram to a particular memory location. The memory location that corresponds to a particular *n*-gram is labeled with the associated language. Once data processing begins, the *n*-grams are sampled from the datastream and used as addresses into memory to discern the language associated with the *n*-gram. The final language is determined by the statistics of the words that appear in each language.

34.4.2 Semantic Processing of TCP Data

Within the intelligence community, there is a need to search through massive amounts of multilingual documents that are encoded using different character sets. It has been shown that computational linguistics and text-processing techniques are effective for sorting through large information sets, extracting relevant documents, and discovering new concepts [33]. There is a problem, however, in that the computational complexity of the text-processing algorithms is such that the document ingest rate is too slow to keep up with the high rate of information flow [34].

To overcome this problem, a system using FPGA hardware was developed for accelerated concept discovery and classification algorithms [35, 36].

Circuits were implemented as reconfigurable hardware modules that dramatically increased data ingest rates. It was found that text analysis algorithms that perform "bag of words" processing were widely used and appropriate for many types of computational linguistics tasks. To investigate the utility of hardware-accelerated text analysis algorithms, a reconfigurable FPGA-based semantic-processing system was developed. The hardware tested a variety of target problems involving concept classification, concept discovery, and language identification [36].

A blend of high-speed network devices and reconfigurable hardware was used to rapidly ingest and process data [35]. Data were received from the network as text or HTML documents and carried over standard TCP/IP packets. The TCP processor decoded the packets that contained the document in one or more TCP/IP input flows. Every word (baseword) in the document was analyzed for its semantic meaning. All words in each document were then counted to determine their frequency of occurrence. A document vector was generated that characterized the document content. It was then scored against a set of vectors that represented known or emerging concepts. Thresholds were used to determine if content could be classified as existing or if a new cluster should be formed.

Figure 34.10 diagrams the dataflow of the semantic-processing system. The FPGAs enabled streaming, computationally intensive semantic-processing functions to be performed in constant time. They performed all of the

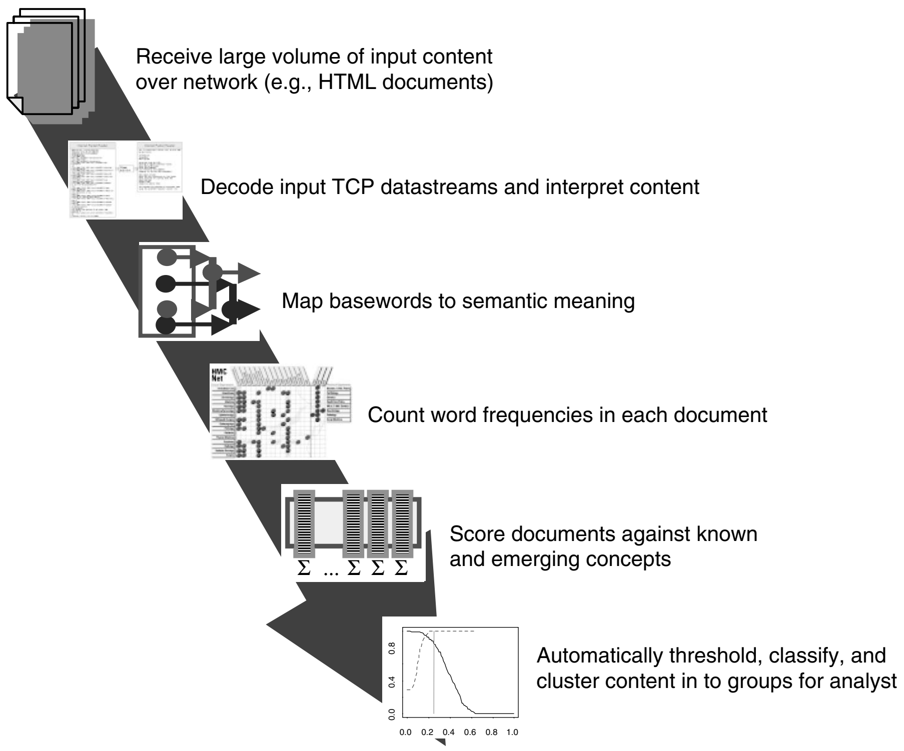
Receive large volume of input content over network (e.g., HTML documents)
Decode input TCP datastreams and interpret content
Map basewords to semantic meaning
Count word frequencies in each document
Score documents against known and emerging concepts
Σ ... Σ Σ Σ
Automatically threshold, classify, and cluster content in to groups for analyst
0.0 0.4 0.8
0.0 0.2 0.4 0.6 0.8 1.0

FIGURE 34.10 ■ Dataflow for the semantic processing system.

data-processing functions for the system shown in the figure except for threshold and classification (which were performed and displayed on a computer console). By using FPGAs to implement all parts of the text processing, the entire system could be dynamically reconfigured to allow variations of algorithms to be evaluated for their content classification or concept-clustering ability. Massive volumes of data were streamed through the system, and the system's precision, recall, throughput, and latency were measured [36].

The RAD circuits on the FPX (shown in Figure 34.3) were used to implement the TCP processor, the baseword module, the count module, the score module, and the report module. All were implemented as modular hardware components on individual FPX platforms connected in a vertical stack. The high-speed network interfaces allowed the FPX platforms to communicate intermediate results of processing to other modules in the system and to send reports to software running on a computer outside the system using standard IP datagrams. Multiple copies of the FPX platform were stacked on each other to implement network intrusion detection and network intrusion prevention. Figure 34.11 is a photograph displaying how five FPX cards were stacked to implement the semantic processing system. Additional modules were added to tag tokens in a context-free grammar [37].

34.5 COMPLETE NETWORKING SYSTEM ISSUES

To deploy complete network systems, additional issues must be considered. First, the hardware must be placed in a form factor appropriate for use in remote network closets. Second, the control and configuration of the hardware must be secure. And third, reconfiguration mechanisms are needed so that entire FPGAs, or (as needed) only parts, can be reconfigured over the network. With dynamic hardware plug-ins, most of the system can remain operational while parts of it are reconfigured. Partial bitfile reconfiguration allows the system itself to remain operational 24 hours a day (which is necessary to maintain a good network uptime) while individual components can still be modified quickly and efficiently. The PARBIT tool allows precompiled partial bitfile configurations to be generated and then quickly deployed into regions of FPGA networking hardware.

34.5.1 The Rack-mount Chassis Form Factor

Networking equipment is typically deployed in the form factor of a chassis that can be mounted into a 19-inch rack. Each unit (U) of a rack is 1.75 inches tall. In a 3U rack-mount chassis, up to four FPX modules could be stacked on each of two ports in the system. Data entered and left the system through the Gigabit Ethernet ports on the front panel. Figure 34.12 is a photograph of FPX modules integrated in a rack-mount chassis.

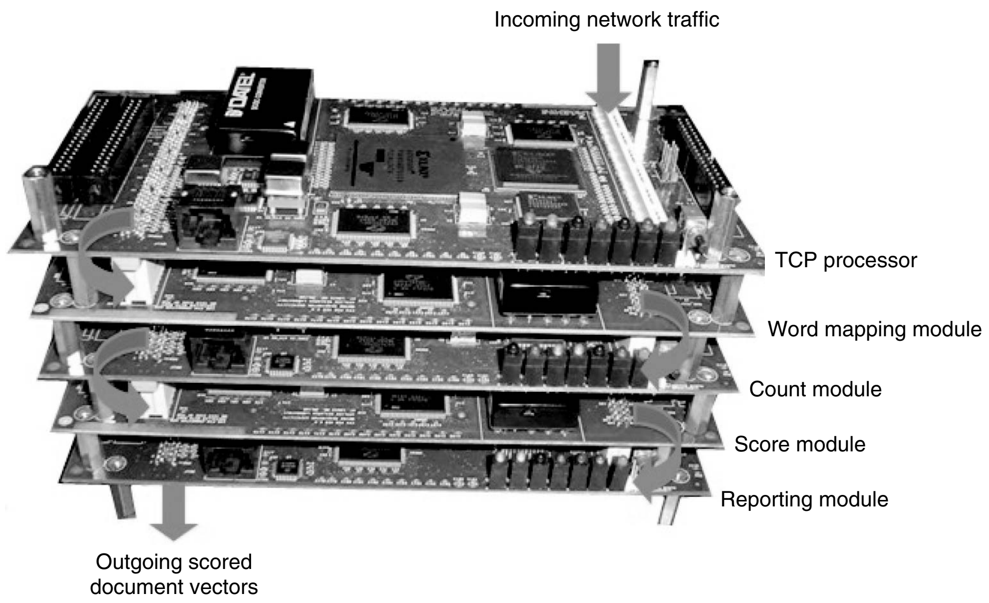

FIGURE 34.11 ■ A stack of the FPX modules implemented the semantic processing system.

FIGURE 34.12 ■ FPX modules integrated in a rack-mount chassis.

34.5.2 Network Control and Configuration

Reconfigurable hardware circuits perform a variety of functions in the networking system. Some parts of the system implemented the infrastructure while others implemented the dynamically reconfigurable logic. Static circuits are used to switch cells between modules. The extensible modules implemented as plug-ins perform the reconfigurable features. The FPX used a combination of statically configured and dynamically configurable logic to implement the complete platform.

On the FPX, the NID was statically configured using a bitfile stored in a PROM. It controlled how data was routed between network modules. and included switching modules that forwarded traffic flows based on virtual paths and circuits found in the ATM cell headers. The NID also contained the logic that enabled other hardware modules to be dynamically loaded over the network. This logic implemented a circuit that used a reliable network protocol to receive full and partial bitfiles over the network. The NID, in turn, buffered this data in a configuration cache and streamed the bitstream into the programming port of the attached FPGA.

The RAD on the FPX was a Xilinx VirtexE-2000E FPGA that received the configuration data and performed application-specific functions implemented as dynamic hardware plug-in (DHP) modules. A DHP consisted of a region of FPGA gates and internal memory bound by the well-defined interface. For bitfiles that used all of the logic on the RAD, the interface was defined by user constraints file (UCF) pins. For partial bitfiles that used less than the entire FPGA, a standard on-chip interface was developed to transmit and receive packet data between modules. A full or partial bitfile was built using standard CAD tools [11].

34.5.3 A Reconfiguration Mechanism

The NID allowed modules created for the FPX platform to be remotely and dynamically loaded into the RAD. This bitstream was sent over the network into the configuration cache, which was implemented by a circuit that controlled an off-chip SRAM. Once a full or partial bitfile was received, a command was sent to the NID to initiate the RAD reconfiguration. On a Xilinx Virtex, the SelectMAP interface loaded a new bitstream into the FPGA. To reprogram the RAD, the NID read the configuration memory and wrote a preprogrammable number of configuration bytes into the RAD FPGA's SelectMAP interface. Figure 34.13 illustrates this process.

The NCHARGE API [9] was developed for debugging, programming, and configuring an FPX. Specifically, it included commands to check the status of an FPX, configure routing on the NID, and perform memory updates and full and partial RAD reprogramming.

NCHARGE provided a mechanism for applications to define their own custom control interface. Control cells were transmitted by NCHARGE and processed by control cell processors (CCPs) on the RAD or NID. To configure routes for the traffic flowing through the system, NCHARGE sent control cells with commands that modified routing tables on the Gigabit switch or on the NID. To check the status of the FPX, NCHARGE sent a control cell to the NID on the FPX, the NID updated fields in the cell, and the software process received the response.

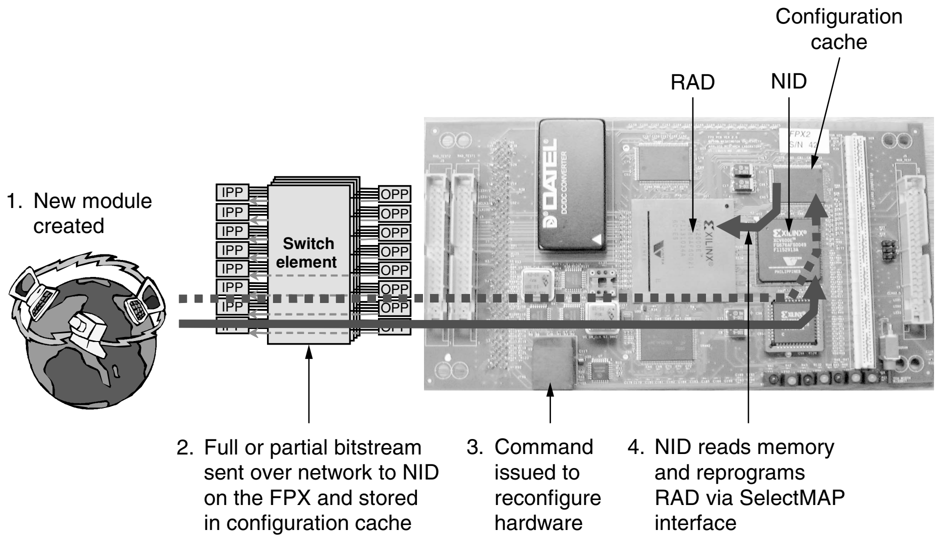
Configuration cache
RAD
NID
1. New module created
IPP
Switch element
OPP
2. Full or partial bitstream sent over network to NID on the FPX and stored in configuration cache
3. Command issued to reconfigure hardware
4. NID reads memory and reprograms RAD via SelectMAP interface

FIGURE 34.13 ■ Remote reconfiguration of the FPX platform.

34.5.4 Dynamic Hardware Plug-ins

Use of runtime reconfiguration in networking systems enables developers of hardware packet-processing applications to achieve a capability similar to that of the dynamically linked libraries (DLLs) used in software applications. Just as a DLL is a software module that can be attached to or removed from a running program as an application demands, DHPs can be loaded into or removed from a running FPGA without disturbing other circuits operating in it. The ability to change the hardware feature set in a running system is particularly useful in packet-processing applications such as firewalls and routers where it is not desirable to suspend the network operation during reprogramming.

A practical system for implementing DHPs was implemented on the FPX and provided sufficient resources for networking, well-defined interfaces to hardware, a complete design methodology, scripts that ran physical implementation tools to place and route logic, and tools that allowed selective reconfiguration of portions of the bitstream. These five elements were analogous to an operating system platform, application programming interface, modular programming methodology, compiler, and linker needed to implement DLLs in the software domain.

34.5.5 Partial Bitfile Generation

Tools and a design methodology were developed to support partial runtime reconfiguration of DHP modules on the FPX platform. The PARBIT tool was developed to transform and restructure bitstreams created by standard computer-aided design tools into partial bitstreams that programmed DHPs.

The methodology allowed the platform to hot-swap application-specific DHP modules without disturbing the operation of the rest of the system [12].

To partially reconfigure an FPGA, it is necessary to isolate a specific area in it and download the configuration only for the bits related to that area. PARBIT transformed and restructured the Xilinx bitstreams to extract and merge data from the bitfile's regions. To restructure the configuration bitfile, it read the original bitfile, a target bitfile, and parameters given by the user that specified the block coordinates of the logic implemented on a source FPGA, the coordinates of the area for a partially programmed target FPGA, and the programming options. After reading these data, PARBIT copied to the target bitstream only the part of the original bitstream related to the area defined by the user.

The target bitstream was used by PARBIT to preserve the part of the configuration data that was in a column specified by the user but outside the partial reconfigurable area. On a Xilinx VirtexE FPGA, the use of the target bitstream was necessary because one reconfiguration frame could span all rows of a column but have a partial reconfigurable area smaller than the column's height. PARBIT allowed arbitrary block regions of a compiled design to be retargeted into any similarly sized region of an FPGA.

To relocate blocks from the original bitfile, a user defined the start and end columns and rows for the block in the original design. Then the user defined where to put this block in a target bitfile of the same device type. The tool generated the partial bitfile containing the area selected by the user (from the original bitfile). This data was used to reconfigure the target device. The configuration bits for the top and bottom input/output blocks (IOBs) from the target device did not change after the partial bitfile was loaded. Those for the columns from the original and target bitfile were merged according to the rows defined by the user.

34.5.6 Control Channel Security

For devices deployed remotely on the Internet, security of the control channel is critical. Remote systems need to be safe from both passive and active network attacks by malicious users. In passive attacks, malicious users glean information by monitoring the system. In active attacks, they attempt to change the system's behavior or paralyze it. Access control mechanisms have been developed to protect remotely configured systems from unauthorized use.

Common attacks include passive eavesdropping, active tampering, replay, and denial of service (DoS). For a passive eavesdropping attack, a malicious user taps the network to copy and analyze its traffic. If the attacker can see clear text control and configuration information, he or she may discover how to control and configure the system. In an active tampering attack, an unauthorized user attempts to gain control of the remote system by issuing bogus control packets. For a replay attack, a malicious user passively captures legitimate traffic and then attempts to change the operation of the system by resending the captured traffic at a later time. For an active DoS attack, the user paralyzes the system by overloading the network with massive amounts of traffic.

Remotely configurable network systems can be made safe by mechanisms that ensure confidentiality of data, provide authentication of the administrator, and guarantee integrity of the messaging. By encrypting messages with the Advanced Encryption Standard (AES) or other secure encryption algorithms, data confidentiality can be protected. With digital signatures generated by public key algorithms, the administrator of the system can be authenticated to guarantee that no one else attempts to modify its operation. The integrity of messages can be ensured by verifying that exactly what is transmitted by the administrator is received by the system. Use of a message authentication code (MAC) can assure users that data are not modified and that no additional control messages are inserted.

The Internet Protocol Security (IPSec) standard provides a mechanism to secure communications across the Internet. Many companies, such as Cisco, have implemented IPSec capability in their networking products. To secure a remotely reconfigurable FPGA, an IPSec in transport mode was designed for a Xilinx Virtex-II Pro FPGA [10]. Security policies at network access points defined who could gain access and under what conditions access was granted. Encryption keys and hash keys remained secret using the security services previously described. The Internet key exchange (IKE) protocol negotiated and exchanged shared secrets between communication entities.

34.6 SUMMARY

As the limits of processor clock scaling are reached, systems that route, process, filter, and transform Internet data scale better in reconfigurable hardware than in software alone. Networking platforms created with FPGA hardware are both fast and flexible. The FPX platform was used to implement over 30 core networking functions.

The combination of Gigabit network interfaces, parallel banks of SRAM and SDRAM, and a large array of reconfigurable logic on the FPX platform enabled it to perform a wide range of networking applications. Modules and protocol wrappers created in reconfigurable hardware were developed on the FPX and provided functionality similar to the procedures and DLLs in software for network processing. Reconfiguration of the modules over the network proved to be as effective for remotely loading new functionality on the FPX as the reprogramming of software on remote PCs.

By using IP wrappers, the FPX platform provided the ability to process ATM cells, AAL5 frames, IP packets, UDP datagrams, and/or TCP/IP flows. Parallel finite automata engines proved useful in detecting regular expressions in packet payloads and TCP traffic flows. Bloom filters that performed parallel hash lookups also proved to be effective for detecting fixed strings in packets and TCP flows. A complete IDS system was implemented that performed a large subset of SNORT using a combination of protocol-processing wrappers, IP header matching circuits, and Bloom filter payload-scanning circuits. A worm and virus detection and blocking system was built using an FPX that demonstrated its utility in providing Internet security.

Reconfigurable hardware holds great promise for new types of networking applications. A language detection circuit was demonstrated that routed traffic based on the language used in a document. A semantic-processing circuit was demonstrated that allowed documents to be classified based on their topic.

Going forward, reconfigurable hardware is becoming the technology of choice for future networking systems. Reconfigurable hardware is the key feature of a new platform, called the NetFPGA. This open platform enables switching and routing of network packets on Gigabit Ethernet links. Because the NetFPGA has many of the same resources as the FPX, it can implement most of the features first prototyped on the FPX [38, 39].

References

[1] J. W. Lockwood. Evolvable Internet hardware platforms. *NASA/DoD Workshop on Evolvable Hardware*, July 2001.

[2] J. W. Lockwood, H. Duan, J. M. Morikuni, S. M. Kang, S. Akkineni, R. H. Campbell. Scalable optoelectronic ATM networks: The iPOINT fully functional testbed. *IEEE Journal of Lightwave Technology*, June 1995.

[3] H. Duan, J. W. Lockwood, S. M. Kang, J. D. Will. A high-performance OC-12/OC-48 queue design prototype for input-buffered ATM switches. *IEEE Infocom '97*, April 1997.

[4] W. Marcus, I. Hadzic, A. McAuley, J. Smith. Protocol boosters: Applying programmability to network infrastructures. *IEEE Communications Magazine* 36(10), 1998.

[5] Wikipedia. OSI model. *http://wikipedia.org/wiki/OSI_model*, July 2006.

[6] F. Braun, J. W. Lockwood, M. Waldvogel. Protocol wrappers for layered network packet processing in reconfigurable hardware. *IEEE Micro* 22(3), February 2002.

[7] D. E. Taylor, J. S. Turner, J. W. Lockwood, T. S. Sproull, D. B. Parlour. Scalable IP lookup for Internet routers. *IEEE Journal on Selected Areas in Communications* 21(4), May 2003.

[8] D. Schuehler, J. W. Lockwood. A modular system for FPGA-based TCP flow processing in high-speed networks. *Proceedings of the 14th International Conference on Field-Programmable Logic and Applications*, August 2004.

[9] T. S. Sproull, J. W. Lockwood, D. E. Taylor. Control and configuration software for a reconfigurable networking hardware platform. *IEEE Symposium on Field-Programmable Custom Computing Machines*, April 2002.

[10] J. Lu, J. W. Lockwood. IPSec implementation on Xilinx Virtex-II Pro FPGA and its application. *Reconfigurable Architectures Workshop*, April 2005.

[11] E. D. Horta, J. W. Lockwood, D. E. Taylor, D. Parlour. Dynamic hardware plugins in an FPGA with partial run-time reconfiguration. *Design Automation Conference*, June 2002.

[12] E. Horta, J. W. Lockwood. Automated method to generate bitstream intellectual property cores for Virtex FPGAs. *Proceedings of the 14th International Conference on Field-Programmable Logic and Applications*, August 2004.

[13] J. W. Lockwood, C. Neely, C. Zuver, D. Lim. Automated tools to implement and test Internet systems in reconfigurable hardware. *SIGCOMM Computer Communications Review* 33(3), July 2003.

[14] J. W. Lockwood, J. Moscola, D. Reddick, M. Kulig, T. Brooks. Application of hardware accelerated extensible network nodes for Internet worm and virus protection. *International Working Conference on Active Networks*, December 2003.

[15] J. Moscola, J. W. Lockwood, R. P. Loui, M. Pachos. Implementation of a content-scanning module for an Internet firewall. *IEEE Symposium on Field-Programmable Custom Computing Machines*, April 2003.

[16] R. Sidhu, V. K. Prasanna. Fast regular expression matching using FPGAs. *IEEE Symposium on Field-Programmable Custom Computing Machines*, April 2001.

[17] R. Franklin, D. Carver, B. L. Hutchings. Assisting network intrusion detection with reconfigurable hardware. *IEEE Symposium on Field-Programmable Custom Computing Machines*, April 2002.

[18] M. Roesch. Snort: Lightweight intrusion detection for networks. *Proceedings of the 13th Administration Conference, LISA*, November 1999.

[19] S. Dharmapurikar, P. Krishnamurthy, T. S. Sproull, J. W. Lockwood. Deep packet inspection using parallel Bloom filters. *IEEE Micro* 24(1), January 2004.

[20] H. Song, S. Dharmapurikar, J. Turner, J. W. Lockwood. Fast hash table lookup using extended Bloom filter: An aid to network processing. *ACM SIGCOMM*, August 2005.

[21] M. Attig, J. W. Lockwood. SIFT: SNORT intrusion filter for TCP. *IEEE Symposium on High Performance Interconnects (Hot Interconnects-13)*, August 2005.

[22] L. Schaelicke, T. Slabach, B. Moore, C. Freeland. Characterizing the performance of network intrusion detection sensors. *Proceedings of the Sixth International Symposium on Recent Advances in Intrusion Detection*, September 2003.

[23] B. Madhusudan, J. W. Lockwood. A hardware-accelerated system for real-time worm detection. *IEEE Micro* 25(1), January 2005.

[24] D. Moore, C. Shannon, G. Voelker, S. Savage. Internet quarantine: Requirements for containing self-propagating code. *IEEE INFOCOM*, 2002.

[25] S. Staniford, V. Paxson, N. Weaver. How to own the Internet in your spare time. *Usenix Security Symposium*, August 2002.

[26] S. Singh, C. Estan, G. Varghese, S. Savage. *The Earlybird System for the Real-time Detection of Unknown Worms*, Technical report CS2003-0761, University of California, San Diego, Department of Computer Science, 2003.

[27] C. Estan, G. Varghese. New directions in traffic measurement and accounting. *ACM SIGCOMM*, August 2002.

[28] C. M. Kastner, G. A. Covington, A. A. Levine, J. W. Lockwood. HAIL: A hardware-accelerated algorithm for language identification. *Proceedings of the 15th Annual Conference on Field-Programmable Logic and Applications*, August 2005.

[29] Global Reach. Global Internet statistics by language. *http://www.glreach.com/globstats/index.php3*, December 2004.

[30] Global Reach. Global Internet statistics: Sources and references. *http://www.glreach.com/globstats/refs.php3*, December 2004.

[31] R. Paulsen, M. Martino. *Word Counting Natural Language Determination*, U.S. Patent 6,704,698, 1996.

[32] J. Schmitt. *Trigram-based Method of Language Identification*, U.S. Patent 5,062,143, 1990.

[33] M. Damashek. *Method of Retrieving Documents that Concern the Same Topic*, U.S. Patent 5,418,951, 1994.

[34] J. B. Sharkey, D. Weishar, J. W. Lookwood, R. Loui, R. Rohwer, J. Byrnes, K. Pattipati, D. Cousins, M. Nicolletti, S. Eick. Information processing at very

high-speed data ingestion rates. In *Emergent Information Technologies and Enabling Policies for Counter Terrosiom*, edited by R. Popp and J. Yin. IEEE Press/Wiley, 2006.

[35] J. W. Lockwood, S. G. Eick, D. J. Weishar, R. Loui, J. Moscola, C. Kastner, A. Levine, M. Attig. Transformation algorithms for datastreams. *IEEE Aerospace Conference*, March 2005.

[36] J. W. Lockwood, S. G. Eick, J. Mauger, J. Byrnes, R. Loui, A. Levine, D. J. Weishar, A. Ratner. Hardware accelerated algorithms for semantic processing of document streams. *IEEE Aerospace Conference,* March 2006.

[37] Y. H. Cho, J. Moscola, J. W. Lockwood. Context-free grammar based token tagger in reconfigurable devices. *Proceedings of the International Workshop on Data Engineering*, April 2006.

[38] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, J. Luo. NetFPGA—An open platform for Gigabit-rate network switching and routing. *IEEE International Conference on Microelectronic Systems Education (MSE2007)*, June 2007.

[39] J. Luo, J. Pettit, M. Casado, N. McKeown, J. W. Lockwood. Prototyping fast, simple, secure switches for ethane. *IEEE Symposium on High-Performance Interconnects (Hot Interconnects-15)*, August 2007.