Automated Tools to Implement and Test Internet Systems in Reconfigurable Hardware

John W. Lockwood, Chris Neely, Chris Zuver, Dave Lim
Applied Research Laboratory
Department of Computer Science and Engineering
Washington University
1 Brookings Drive, Campus Box 1045
Saint Louis, MO 63130
lockwood@arl.wustl.edu
http://www.arl.wustl.edu/arl/projects/fpx

ABSTRACT

Tools have been developed to automatically integrate and test networking systems in reconfigurable hardware. These tools dynamically generate circuits for Field Programmable Gate Arrays (FPGAs). A library of hardware-accelerated modules has been developed that processes Internet Protocol (IP) packets, performs header rule matching, scans packet payloads, and implements per-flow queueing. Other functions can be added to the library as extensible modules.

An integration tool was developed to enable a network administrator to specify how a customized system should examine, drop, buffer, and/or modify packets. This tool joins together modules from the library to create a composite circuit that performs multiple functions. The tool allows additional modules to be quickly added to the library and integrated into systems. The integration tool has been used to create circuits that perform Internet firewall, network intrusion detection, network intrusion prevention, and Denial of Service (DoS) attack protection functions.

A test tool was developed to automatically verify that circuits created by the integration tool run properly in reconfigurable hardware. Circuits created by the integration tool are deployed into a Field-programmable Port Extender (FPX) platform. As new modules were added to the library, the test tool reconfigured the logic on the FPX, injected traffic, and monitored the resulting packets.

By using hardware, not software, networking system can process millions of packets per second. Together, the integration and test tools simplify the otherwise difficult task of developing reconfigurable hardware for networking systems and testing them at Gigabit per second rates.

Categories and Subject Descriptors

C.2.1 [**Computer Systems Organization**]: Computer-Communication Networks—*Network Architecture and Design*; B.4.3 [**Hardware**]: Input/Output and Data Communications—*Interconnections (Subsystems)*; B.7.1 [**Hardware**]: Circuits—*VLSI*; B.7.2 [**Hardware**]: Circuits—*Design Aids*

General Terms

Networks, tools, reconfigurable hardware

Keywords

Field Programmable Gate Array (FPGA), Internet, firewall, network intrusion detection and prevention

1. INTRODUCTION

Through the research on extensible networks, mechanisms have been developed that allow Internet routers to dynamically load new features over a network [1]. These mechanisms enabled a router to perform new functions even after it was deployed in the field. Currently, it is mostly software that is loaded into a processor to implement new functionality [2]. The bottleneck of processing packets in software can make the system unusable for all but niche applications. By using reconfigurable hardware, it is possible to deploy extensible systems that can dynamically load new features and achieve high levels of performance.

1.1 Extensible Networks

Several types of reprogrammable systems have been proposed or developed that use load software over a network to provide extensibility. Most systems perform computation with microprocessors. Some use Java to execute a code capsule carried within a packet, while others execute binary code [3]. Some newer systems use *Network Processors* to achieve higher performance by running code on a few (6-32) parallel RISC cores [4]. While these software-based systems have outstanding flexibility, their packet processing functionality is still limited due to sequential execution of instructions in software.

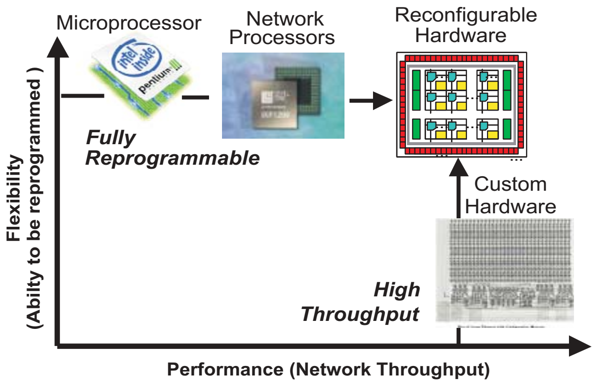


Figure 1: Reconfigurable hardware leverages performance advantage of custom hardware with flexibility of to reprogram like software

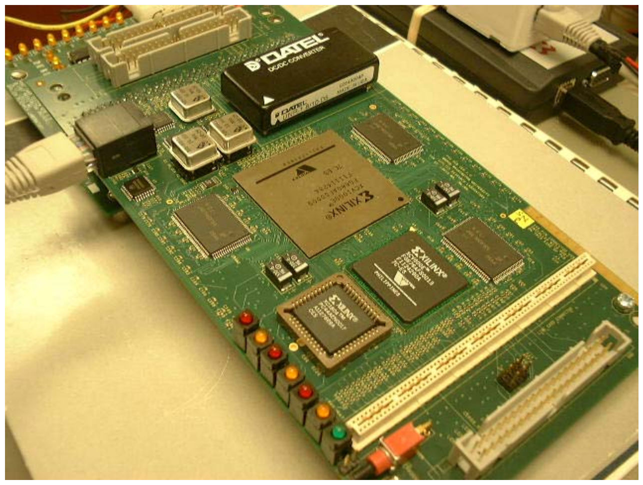


Figure 2: The FPX Platform

1.2 The Need for Speed

In order to process large amounts of data, custom hardware can be used to process packets. Today's fastest firewalls and routers achieve their high throughput because they perform packet processing operations in custom silicon or Application Specific Integrated Circuits (ASICs). Such systems contain thousands of parallel logic circuits and finite state machines optimized to switch, route, filter, queue, and process Internet datagrams in hardware [5]. While ASICs and custom silicon networking chips have high performance, they offer little or no programmability. It is difficult to build effective network intrusion detection and prevention systems in underlying technology that lacks flexibility.

1.3 Advantages of Reconfigurable Hardware

Field Programmable Gate Arrays (FPGAs) offer a way to combine the advantages of software flexibility and hardware performance. On a FPGA, logic gates and the wires that interconnect the logic can be reconfigured to perform arbitrary functions. FPGAs can be reconfigured either at compile-time or at run-time [6]. Compile-time reconfigurable systems load the FPGA just once when the system is started. Run-time reconfigurable systems can dynamically reprogram hardware even while the system is running [7] [8] [9] [10] [11].

Figure 1 illustrates how components found in networking systems trade off flexibility and performance. The microprocessor excels at flexibility because the functions that it implements can simply changed by loading new software into memory. The performance, however, is limited because the processor executes instructions sequentially. A superscalar processor performs slightly better because it can execute a few instructions in parallel. Network processors offer somewhat better performance than microprocessors because they contain a few (6-32) processors that can run in parallel [4]. Custom hardware achieves the highest level of performance because it can implement thousands of parallel logic functions in hardware. In networking systems, most custom hardware is implemented with ASICs. The flexibility of an ASIC, however, is quite limited because the logic and interconnect cannot be modified after fabrication. FPGAs excel at both flexibility and performance because they are fully reconfigurable and because they can achieve high performance.

While it is true that the processing performance of a FPGA is less than that of an ASIC due to the addition of the programmable control circuits, this penalty is often offset in practice because FPGA-based products can get to market much faster than ASICs.

1.4 Looking Ahead

Reprogrammable logic will continue to expand in capability and improve in performance. In 2003, FPGAs were one of the first commercial products to be mass produced with 0.09μm (90 nanometer) silicon process technology. The FPGA of 2005 is predicted to be implemented with a 0.07μm (70 nanometer) silicon process technology and have two billion transistors [12]. With such high levels of integration, the incremental cost to add extra FPGA gates becomes insignificant as compared to the cost to design, implement, fabricate, and deploy a complete network system. Going forward, it is likely that the latest innovations in silicon process technology will continue to be available in FPGAs before ASICs. FPGA vendors can leverage volume production of standard parts to justify the large cost of adopting new technology.

2. THE FPX PLATFORM

An open networking hardware platform called the Field Programmable Port Extender (FPX) [13] has been developed that enables the rapid prototype and deployment of packet processing modules in reprogrammable hardware [14]. A photograph of the FPX platform is shown in Figure 2. The FPX implements all logic using two FPGA devices: the Network Interface Device (NID) and the Reprogrammable Application Device (RAD). These FPGAs, which perform all of the computation on the FPX, are the two largest Xilinx chips located in the center of the FPX platform. The FPX also contains five parallel banks of off-chip memories. Three Parallel banks of pipelined Static RAM (SRAM) enable the FPX to implement high-speed table lookup operations and cache reconfiguration data. Two parallel banks of Synchronous DRAM (SDRAM) enable to the FPX to perform per-flow queuing and buffer large amounts of data.

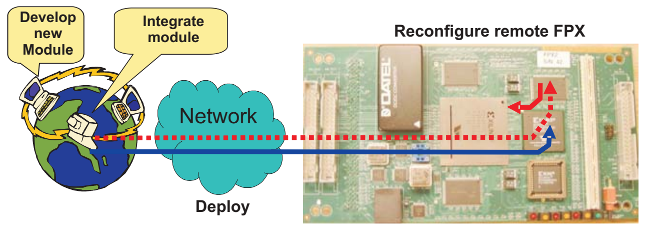


Figure 3: The FPX platform can be reconfigured over a network

The FPGAs were chosen so that the FPX platform could both implement all of the core network functions needed by current firewalls and network intrusion systems yet still have sufficient capacity to load additional features. The NID is implemented with a Xilinx Virtex 600E FPGA, which is configured when the FPX is powered on to implement the core operations of the FPX. It contains logic to implement per-flow routing and process reconfiguration commands. The RAD is implemented with a Virtex 2000E FPGA, which contains the FPGA equivalent of two million ASIC gates, and can be dynamically reconfigured at run-time over the network using reconfiguration control circuits on the NID.

2.1 Extensible Features

The key feature of the FPX relates to the mechanism by which it can be reloaded with new modular hardware components over the network. A diagram that shows how the FPX can be reconfigured is shown in Figure 3. When a new circuit for the RAD is ready to be deployed, the bitfile for that circuit is sent over the network and stored in a SRAM cache attached to the NID. Once the network administrator is ready to use that circuit, a command is sent to instruct the FPX to dynamically reconfigure some or all of the logic on the RAD.

The FPX supports both full reconfiguration and partial reconfiguration of the RAD. Partial reconfiguration is supported by generating a FPGA bitfile that contains only the portion of the FPGA that changed. The NID can write only those selected reconfiguration frames to the RAD's *SelectMap* reprogramming port [15]. This feature enables other module(s) on the RAD to continue processing packets while other parts of the circuit are being reconfigured.

The method to perform partial reconfiguration of the FPX is similar to that of [16] [17] in that only a portion of the device is reconfigured. The method differs, however, in that the controller which reconfigures regions of the RAD is implemented in hardware rather than software. The use of the hardware method allows the FPX to reconfigure in milliseconds rather than the several seconds required by other techniques [18].

3. INTEGRATION TOOL

An integration tool was developed to automate the generation of complete networking systems. The integration tool first generates synthesizable VHDL code then compiles it into a bitfile that can loaded into a RAD. Complete systems consist of multiple modular hardware components joined together to perform a composite set of functions [19].

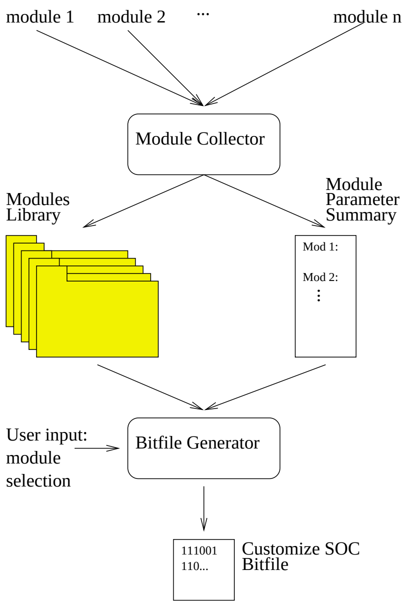


Figure 4: Integration Server

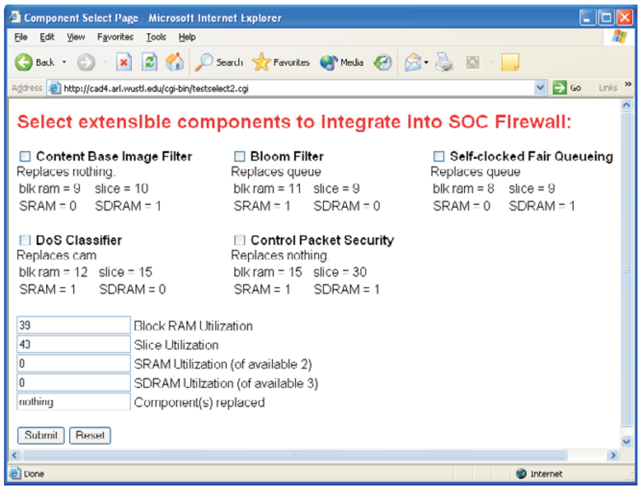
The integration tool allows changes to be made to any module without affecting other parts of the system. To do this, the integration server acts as a repository for all modules. When a developer updates or creates a new module, that module is uploaded to the integration server, and the integration tool is used to add that module to the library. When a network administrator needs a new system, a customized bitfile is automatically generated by the integration server that contains the new set of modules desired.

Modules on the RAD are implemented in hardware, not software. Each module has input and output ports that are used to connect it to the rest of the system. The integration server connects these ports to ports on other modules and/or to ports on the top-level design. If done manually, this task would be time-consuming and error-prone. The integration server automates this process.

The integration server is divided into two logical parts: the module collector and the bitfile generator. Each part is controlled via a web-based user interface. The collector reads newly created modules and stores them in the modules database. The bitfile generator creates a bitfile of the complete system using a subset of the modules in the database. A diagram of the integration server is shown in Figure 4

3.1 Module Collector

When a module is ready to be integrated into the RAD, the module collector reads the uploaded top-level VHDL and EDIF files which describe that module. The module collector then verifies that the uploaded files are of the correct format. If the file formats are valid, the collector parses the VHDL file to identify all of the ports on the module. For each interface, the collector prompts the developer to identify a corresponding interface on the top-level system design. Next, the collector prompts the developer to determine how each port should be mapped to each interface.


Component Select Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://cad4.arl.wustl.edu/cgi-bin/testselect2.cgi
Go Links
Done
Internet

Select extensible components to integrate into SOC Firewall:

☐ **Content Base Image Filter**
Replaces nothing.
blk ram = 9 slice = 10
SRAM = 0 SDRAM = 1

☐ **Bloom Filter**
Replaces queue
blk ram = 11 slice = 9
SRAM = 1 SDRAM = 0

☐ **Self-clocked Fair Queueing**
Replaces queue
blk ram = 8 slice = 9
SRAM = 0 SDRAM = 1

☐ **DoS Classifier**
Replaces cam
blk ram = 12 slice = 15
SRAM = 1 SDRAM = 0

☐ **Control Packet Security**
Replaces nothing.
blk ram = 15 slice = 30
SRAM = 1 SDRAM = 1

39	Block RAM Utilization
43	Slice Utilization
0	SRAM Utilization (of available 2)
0	SDRAM Utilzation (of available 3)
nothing	Component(s) replaced

Submit Reset

Figure 5: A web-based interface is used by network administrators to generate a custom systems

Once all parameters of a newly created module are specified, the module collector records that information. To ensure that the module works as claimed, it generates a bitfile that with just that new module integrated into a baseline system. It synthesizes the circuit and next places and routes that circuit into a FPGA. The resulting bitfile is then sent back to the developer for final testing with just that new module configured within a baseline system. Lastly, the new module is next added to the *modules library*, and the corresponding parameters are stored in a *Module Summary File* (MSF).

3.2 Bitfile Generator

To allow a network administrator to easily generate a customized RAD circuit, the *bitfile generator* parses the MSF created by the *bitfile collector* and displays a menu of available system features. Although the resources on an FPGA are large, they are not infinite. As such, the bitfile generator includes code to ensure that the modules selected for integration will actually fit into the resources available on the target FPGA. For each module selected, the generator automatically recomputes the available resources. Once the selections are made, the generator prepares to build a bitfile with the selected subset of modules. It first verifies that the resource utilization of the selected modules combined with the baseline system does not exceed the available resources on the target FPGA. In the case of the FPX, the integration tool ensures that the requested circuit does not exceed the available number of logic slices, on-chip BlockRAMs, interfaces off-chip SRAM, and interfaces to off-chip SDRAM. The module collector then writes a new structural description of the circuit and invokes hardware synthesis tools to generate a bitfile for testing the module.

The web form used to generate an extensible network system is shown in Figure 5. Every time the administrator requests this page, the bitfile generator rebuilds the page contents by parsing the MSF, and displays a list of all of the available modules. The bitfile generator tracks the device utilization and displays it at the bottom of the form. In this screen shot, five modules are available for integration. With none of the extensible modules selected, 30% of the block RAMs are utilized and 43% of the logic slices are used. As modules are selected, the values corresponding to the available resources are recomputed.

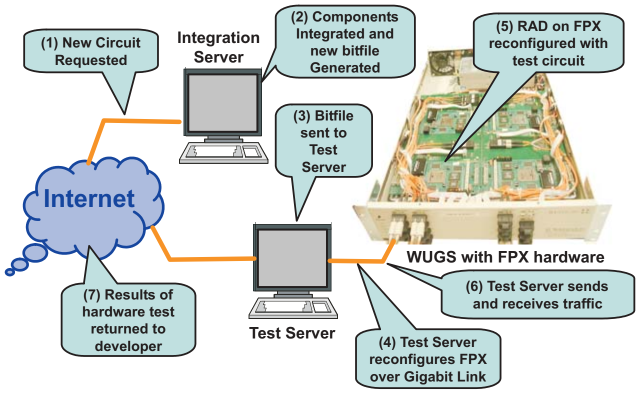

(1) New Circuit Requested
Integration Server
(2) Components Integrated and new bitfile Generated
(5) RAD on FPX reconfigured with test circuit
(3) Bitfile sent to Test Server
Internet
WUGS with FPX hardware
(6) Test Server sends and receives traffic
(7) Results of hardware test returned to developer
Test Server
(4) Test Server reconfigures FPX over Gigabit Link

Figure 6: Typical use of the Integration and Testing Tools

3.3 Design Flow

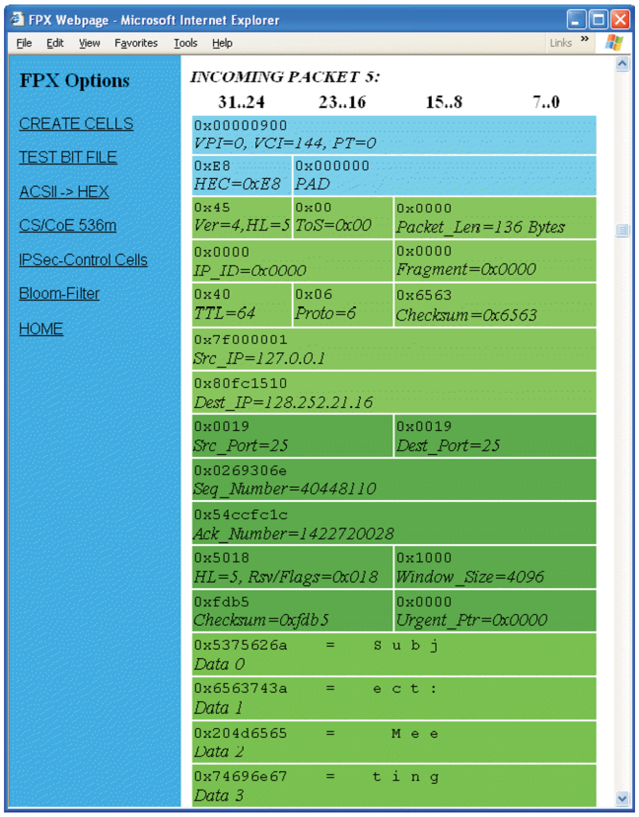
A design flow is used to synthesize the RAD circuit. The process of building a new module involves compiling the logic, verifying the functionality of the circuit, synthesizing the circuit to gates, placing constraints on the location of the modules and I/O ports, placing and routing the circuit, generating a bitfile, uploading the bitfile to the FPX for in-system testing, testing the operation of the module with Internet traffic, and lastly verifying that the packets are properly processed.

Hardware synthesis is performed with Synplicity's Amplify tool. The resulting EDIF netlist is generated is then fed into the Xilinx backend FPGA design flow. Pin locations specified in a constraint file are used to map the pins of the RAD to appropriate I/O pins of the Xilinx Virtex XCV2000E part. Next, the Xilinx place and route tools are run to implement the FPGA circuit. The content of the resulting bitstream is then sent to the FPX for at-speed testing. The total time required to iterate from a change in the VHDL source code to in-system testing is under 10 minutes.

4. NETWORK TEST TOOL

A test tool was implemented that allows batches of RAD circuits to be evaluated on the FPX platform [20]. Once a bitfile has been submitted to the test server, it is added to a queue of configurations waiting to be tested. The queue processes bitfiles in the order in which they are received. The server automatically sets up routes through the network in order to forward traffic through the FPX. A diagram that shows the interaction of the integration and test server is shown in Figure 6.

The test server was created using a combination of Hypertext Markup Language (HTML) and Common Gateway Interface (CGI) scripts. The hardware test server runs on a PC running NetBSD operating systems and uses the NCHARGE software to interface with the FPX platform [21]. A queue of jobs is maintained to ensure that only one bitfile is programmed into the FPX at a time. Once all of the IP traffic



INCOMING PACKET 5:

31..24	23..16	15..8	7..0
0x00000900 *VPI=0, VCI=144, PT=0*			
0xE8 *HEC=0xE8*	0x000000 *PAD*		
0x45 *Ver=4,HL=5*	0x00 *ToS=0x00*	0x0000 *Packet_Len=136 Bytes*	
0x0000 *IP_ID=0x0000*		0x0000 *Fragment=0x0000*	
0x40 *TTL=64*	0x06 *Proto=6*	0x6563 *Checksum=0x6563*	
0x7f000001 *Src_IP=127.0.0.1*			
0x80fc1510 *Dest_IP=128.252.21.16*			
0x0019 *Src_Port=25*		0x0019 *Dest_Port=25*	
0x0269306e *Seq_Number=40448110*			
0x54ccfc1c *Ack_Number=1422720028*			
0x5018 *HL=5, Rsv/Flags=0x018*		0x1000 *Window_Size=4096*	
0xfdb5 *Checksum=0xfdb5*		0x0000 *Urgent_Ptr=0x0000*	
0x5375626a = s u b j *Data 0*			
0x6563743a = e c t : *Data 1*			
0x204d6565 = M e e *Data 2*			
0x74696e67 = t i n g *Data 3*			

Figure 7: View of packets displayed on test server

required to test a module has been processed by the RAD, the bitfile on the RAD is reconfigured to contain the next job in the queue. Most tests complete in less than 15 seconds.

To make interpretation of the packets that result from processing by the RAD easier for a human to interpret, the fields of the IP headers are parsed, labelled, and color coded. The packets that return from the FPX are then displayed as an HTML document on the test server, as shown in Figure 7.

5. NETWORK INTRUSION SYSTEMS

As the Internet has grown, demand for network security has significantly increased. Internet hosts are continuously the target of attacks from machines located around the world. Firewalls and network intrusion prevention systems can help protect hosts by selectively filtering malicious traffic.

Internet firewalls thwart attacks by filtering packets using rules based on the values of headers. Network intrusion detection and prevention systems scan entire packet payloads to look for signatures of well-known attacks. Systems like *SNORT* passively scan the payloads of packets for specific signatures [22]. Systems such like Hogwash actively drop or sanitize packets that appear to be malicious.

Denial of Service (DoS) or Distributed DoS (DDoS) attacks occur when a remote machine or group of machines flood traffic to a victim host at high rates. To defend internal networks from a DoS or DDoS attack, systems can filter or rate-limit malicious traffic flows.

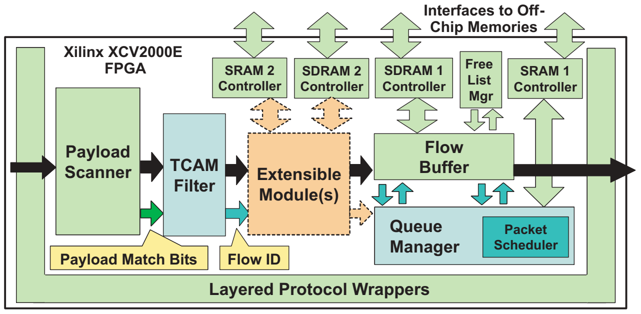


Figure 8: Architecture of baseline RAD circuit that contains core components of the Network Intrusion Prevention System

5.1 Core Components

To simplify the construction of complex Internet packet processing systems, a library of modular networking hardware components was developed. The baseline system includes a set of layered protocol wrappers to process Internet packets; Ternary Content Addressable Memories (TCAMs) to process header rules; a packet payload scanner to detect regular expressions that appear in traffic flows; a flow buffer that uses off-chip SDRAM to store tens to hundreds of megabytes of data; and a queue manager that performs packet scheduling for individual traffic flows.

The existing components can be integrated to protect networks from many types of existing attacks. In order to protect against future threats, the system provides a mechanism to insert extensible modules. The additional features are implemented as components in reconfigurable hardware [23].

The top-level architecture of the SoC firewall is shown in Figure 8. When data first enters the system, a set of layered protocol wrappers reassembles data and identifies the location of the headers. Next, a packet payload scanner searches the content of the packets to find keywords specified as regular expressions. After scanning the packet, the values of headers are compared to a set of rules stored in TCAM filters. Some rules can cause the TCAM filters to outright drop packets, while other rules can assign the packet to a low-priority flow. After classification, the queue manager schedules packets for transmission from the flow buffer. To enable buffering of tens to hundreds of megabytes of data, off-chip SDRAM is used to store packets. Once a packet is scheduled for transmissions, it is read from SDRAM by the flow buffer and transmitted out of the firewall. Details of each module are discussed in the subsections that follow.

5.2 Protocol Processing

The layered protocol wrappers consist of VHDL-specified components that process high-level Internet protocols directly in hardware. The baseline wrappers consist of four components that are used together to transmit and receive: fixed-length cells, variable-length frames, Internet Protocol (IP) packets, and User Datagram Procotol (UDP) frames [24]. An additional wrapper has been developed to process a Transmission Control Protocol (TCP) flow [25] as well as multiple TCP/IP traffic flows [26].

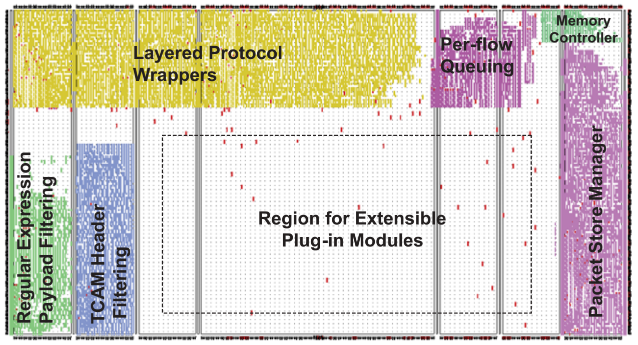

Layered Protocol Wrappers
Per-flow Queuing
Memory Controller
Regular Expression Payload Filtering
TCAM Header Filtering
Region for Extensible Plug-in Modules
Packet Store Manager

Figure 9: Layout of the Baseline Network Intrusion Prevention System in the Xilinx Virtex XCV2000E that implements the RAD on the FPX

5.3 Payload Processing

The payload scanner allows packets to be classified based using keywords or regular expressions that appear anywhere within a packet. Traffic flows that contain phrases like 'MAKE MONEY FAST', 'CALL NOW', and 'Limited Time Offer', for example, can be classified as SPAM by the payload scanner. Traffic flows that contain signatures for Internet worms and computer viruses can be classified as malicious [27]. A traffic flow that contains a MIME-64 encoded attachment infected with the *SoBigF* Internet worm, for example, can be identified by detecting that the payload contains "HEX(683063423739)".

Regular expressions allow content to be specified with wildcard characters (specified by '?') or strings of multiple characters (specified by '*'). The regular expression "A—albert ? E—einstein" matches all four case variations of the name Albert Einstein and allows the middle initial to be an arbitrary character. To scan strings with high throughput, a design flow was created to automatically generate a set of parallel Deterministic Finite Automata (DFA) that scan for each regular expression. A match is detected when the sequence of arriving bytes causes a DFA to reach a matching state [28]. Finite automata can also be created to modify the content of the traffic, thus performing the FPGA equivalent of the common Unix Stream Editor (SED) [29].

5.4 Flow Buffering

To provide Quality of Service (QoS) for traffic that passes through the network, the queue manager performs both class-based and per-flow queuing. Class-based queuing allows certain types of traffic to receive better service than other traffic. Per-flow queuing ensures that no single traffic consumes an unfair amount of bandwidth.

To support the multiple classes of service, traffic flows are organized by the queue manager in four classes. Traffic from flows in higher-priority classes are transmitted before traffic from lower-priority classes. Multiple linked lists of packets are managed to support per-flow queuing. All management of queues and tracking of free memory space is computed in the FPGA hardware using constant-time, linked-list data structures maintained in SRAM. The queue manager includes circuits to enqueue traffic flows, dequeue traffic flows, and to schedule flows for transmission.

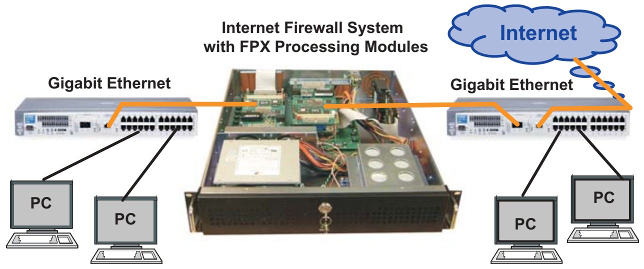


Figure 10: Typical deployment of an FPX-based system in a Gigabit Ethernet network

Within the scheduler, four separate queues of flows are maintained (one for each class). Within each class of traffic, the queue manager performs round-robin queuing of individual flows. When the first packet of a flow arrives that has no packets already buffered, the flow identifier is inserted into a scheduling queue for that packet's class of service and the flow state table is updated. When another packet of a flow that is already scheduled arrives, the packet is simply appended to the linked list [30]

5.5 Device Layout

The core components of the system, including the layered protocol wrappers, TCAM packet filters, regular expression matching engines, per-flow queueing manager, and packet store manager with the SDRAM controller were synthesized into the RAD of the FPX. Device Utilization Placement was constrained using Synplicity's Amplify tool to lock the location of modules into specific regions of the FPGA. A view of the placed and routed Xilinx Virtex XCV2000E is shown in Figure 9 [23]. Note that the center region of the chip was left empty for insertion of extensible modules.

5.6 Testing

A complete system with the protocol wrappers, CAM filter, flow buffer, and queue manager were synthesized and operated at 62.5 MHz on the Xilinx Virtex XCV2000E-6. Each of these components process 32 bits of data in every cycle, thus giving the system a throughput of 32*62.5MHz = 2 Gigabits/second. The regular expression scanning circuit for the list of SPAM keywords was synthesized at 37 MHz [28]. Given that each FSM processes 8 bits of data per cycle, the throughput of the SPAM filter is 8*37MHz=296 Megabits/second per FSM. By running 8 regular expression scanners in parallel, the throughput of the payload matching circuit achieves 8*8*37MHz=2.4 Gigabits/second.

The system has been used to process live network traffic in several configurations. Different line cards allow the FPX to attach to both Asynchronous Transfer Mode (ATM) networks and Gigabit Ethernet networks. In one ATM configuration, FPX cards can be placed on each of the eight ports of the Washington University Gigabit Switch (WUGS) [5]. In another ATM configuration, the FPX can be placed in a small chassis and connected to a pair of OC line cards. In the Gigabit Ethernet configuration, the FPX is placed into a chassis and connected to a pair of Gigabit Ethernet line cards. Traffic sent between the Gigabit Ethernet Switches pass through the FPX in the chassis, as shown in Figure 10.

6. CONCLUSIONS

By processing network traffic in reconfigurable hardware, extensible networking systems can achieve both flexibility and high performance. Using the Field-programmable Port Extender (FPX), an extensible networking system has been developed that implements all functionality as modular components in reconfigurable hardware. New features are dynamically loaded into remote FPX systems by sending commands over the network. The use of parallel hardware circuits enable the FPX to scan Internet traffic at multi-Gigabit per second rates.

Developers have used the tools to integrate and test new networking modules in reconfigurable hardware. The tools have web-based user interfaces that them to operate on-line. The integration tool enables a hardware developer to upload a new module and specify exactly how it fits into a system. The integration tool, in turn, stores this information into an on-line library.

Network administrator can browse the library and select which features they need for their network. Once a set of features is selected, the integration tool dynamically generates a customized network processing circuit. A test tool was developed that verifies that circuits operate properly in hardware. To perform a test, the tool reconfigures a FPX, transmits Internet Protocol packets to the hardware, and analyzes the traffic that is returned.

The FPX platform developed at Washington University has been used to create Internet firewalls, network intrusion detection, and network intrusion prevention systems in hardware. The tools described in this paper enable hardware-accelerated, networking systems to be much more easily integrated and tested in reconfigurable hardware.

7. ACKNOWLEDGEMENTS

The authors wish to thank Todd Sproull for developing the NCHARGE control software, Edson Horta for his development of tools that support partial reconfiguration, Sarang Dharmapurikar for his implementation of the SDRAM flow buffer, James Moscola for his work on the regular expression matching circuit, David Taylor for his work on Dynamic Hardware Plugins (DHPs), David Parlour of Xilinx for his help with the design of DHPs for the Virtex, Jon Turner for his assistance in design of extensible networks, John Wroclawski for his thoughtful reviews of an earlier draft of the paper, the anonymous reviewers of this SIGCOMM CCR issue for their helpful comments, and all of the visitors, graduate researchers, and CSE536 and CS536 students that have developed modules for the FPX platform and provided valuable feedback.

8. REFERENCES

[1] D. L. Tennenhouse et al., "A Survey of Active Network Research," in *IEEE Communications Magazine*, pp. 80–86, Jan. 1997.

[2] D. Decasper, G. Parulkar, S. Choi, J. DeHart, T. Wolf, and B. Plattner, "A scalable high-performance active network node," in *IEEE Network, Vol. 13, No. 1*, January/February 1999.

[3] D. J. Wetherall, J. V. Guttag, and D. L. Tennenhouse, "ANTS: a toolkit for building and dynamically deploying network protocols," in *IEEE Conference on Open Architectures and Network Programming (OPENARCH)*, Apr. 1998.

[4] T. Wolf and J. Turner, "Design issues for high performance active routers," in *Proceedings of International Zurich Seminar on Broadband Communications*, (Zurich, Switzerland), February 2000.

[5] J. Turner, T. Chaney, A. Fingerhut, and M. Flucke, "Design of a Gigabit ATM Switch," in *In Proceedings of Infocom 97*, Mar. 1997.

[6] S. Hauck, "The roles of FPGAs in reprogrammable systems," *Proceedings of the IEEE*, vol. 86, pp. 615–638, Apr. 1998.

[7] S. McMillan and S. Guccione, "Partial run-time reconfiguration using JRTR," in *Field-Programmable Logic and Applications (FPL)*, (Villach, Austria), pp. 352–360, Aug. 2000.

[8] B. L. Hutchings and M. J. Wirthlin, "Implementation approaches for reconfigurable logic applications," in *Field-Programmable Logic and Applications (FPL)* (W. Moore and W. Luk, eds.), (Oxford, England), pp. 419–428, Springer-Verlag, Berlin, Aug. 1995.

[9] D. Ross, O. Vellacott, and M. Turner, "An FPGA-based Hardware Accelerator for Image Processing," in *More FPGAs: Proceedings of the 1993 International workshop on field-programmable logic and applications* (W. Moore and W. Luk, eds.), (Oxford, England), pp. 299–306, 1993.

[10] H. Fallside and M. J. S. Smith, "Internet connected FPL," in *Field-Programmable Logic and Applications (FPL)*, (Villach, Austria), pp. 48–57, Aug. 2000.

[11] H. Schmit, "Incremental reconfiguration for pipelined applications," in *IEEE Symposium on FPGAs for Custom Computing Machines* (K. L. Pocek and J. Arnold, eds.), (Los Alamitos, CA), pp. 47–55, IEEE Computer Society Press, 1997.

[12] P. Alfke, "Advancements in designing with FPGAs," in *DesignCon*, (Santa Clara, CA), Jan. 2001.

[13] J. W. Lockwood, J. S. Turner, and D. E. Taylor, "Field programmable port extender (FPX) for distributed routing and queuing," in *ACM International Symposium on Field Programmable Gate Arrays (FPGA'2000)*, (Monterey, CA, USA), pp. 137–144, Feb. 2000.

[14] J. W. Lockwood, N. Naufel, J. S. Turner, and D. E. Taylor, "Reprogrammable Network Packet Processing on the Field Programmable Port Extender (FPX)," in *ACM International Symposium on Field Programmable Gate Arrays (FPGA'2001)*, (Monterey, CA, USA), pp. 87–93, Feb. 2001.

[15] E. Horta and J. W. Lockwood, “PARBIT: a tool to transform bitfiles to implement partial reconfiguration of field programmable gate arrays (FPGAs),” Tech. Rep. WUCS-01-13, Washington University in Saint Louis, Department of Computer Science, July 6, 2001.

[16] W. Westfeldt, “Internet reconfigurable logic for creating web-enabled devices.” Xilinx Xcell, Q1 1999.

[17] S. Kelem, “Virtex configuration architecture advanced user’s guide.” Xilinx XAPP151, Sept. 1999.

[18] E. L. Horta, J. W. Lockwood, D. E. Taylor, and D. Parlour, “Dynamic hardware plugins in an FPGA with partial run-time reconfiguration,” in *Design Automation Conference (DAC)*, (New Orleans, LA), June 2002.

[19] D. Lim, C. E. Neely, C. K. Zuver, and J. W. Lockwood, “Internet-based tool for system-on-chip integration,” in *International Conference on Microelectronic Systems Education (MSE)*, (Anaheim, CA), June 2003.

[20] C. E. Neely, C. K. Zuver, and J. W. Lockwood, “Internet-based tool for system-on-chip project testing and grading,” in *International Conference on Microelectronic Systems Education (MSE)*, (Anaheim, CA), June 2003.

[21] T. Sproull, J. W. Lockwood, and D. E. Taylor, “Control and configuration software for a reconfigurable networking hardware platform,” in *IEEE Symposium on Field-Programmable Custom Computing Machines, (FCCM)*, (Napa, CA), Apr. 2002.

[22] M. Roesch, “SNORT - lightweight intrusion detection for networks,” in *LISA ’99: USENIX 13th Systems Administration Conference*, (Seattle, Washington), Nov. 1999.

[23] J. W. Lockwood, C. Neely, C. Zuver, J. Moscola, S. Dharmapurikar, and D. Lim, “An extensible, system-on-programmable-chip, content-aware Internet firewall,” in *Field Programmable Logic and Applications (FPL)*, (Lisbon, Portugal), p. 14B, Sept. 2003.

[24] F. Braun, J. W. Lockwood, and M. Waldvogel, “Layered protocol wrappers for Internet packet processing in reconfigurable hardware,” in *Proceedings of Symposium on High Performance Interconnects (HotI’01)*, (Stanford, CA, USA), pp. 93–98, Aug. 2001.

[25] D. V. Schuehler and J. Lockwood, “TCP-Splitter: A TCP/IP flow monitor in reconfigurable hardware,” in *Hot Interconnects*, (Stanford, CA), pp. 127–131, Aug. 2002.

[26] D. V. Schuehler, J. Moscola, and J. W. Lockwood, “Architecture for a hardware based, tcp/ip content scanning system,” in *Hot Interconnects*, (Stanford, CA), pp. 89–94, Aug. 2003.

[27] J. W. Lockwood, J. Moscola, M. Kulig, D. Reddick, and T. Brooks, “Internet worm and virus protection in dynamically reconfigurable hardware,” in *Military and Aerospace Programmable Logic Device (MAPLD)*, (Washington DC), p. E10, Sept. 2003.

[28] J. Moscola, J. Lockwood, R. P. Loui, and M. Pachos, “Implementation of a content-scanning module for an Internet firewall,” in *FCCM*, (Napa, CA), Apr. 2003.

[29] J. Moscola, M. Pachos, J. W. Lockwood, and R. P. Loui, “Implementation of a streaming content search-and-replace module for an internet firewall,” in *Hot Interconnects*, (Stanford, CA, USA), pp. 122–129, Aug. 2003.

[30] H. Duan, J. W. Lockwood, S. M. Kang, and J. Will, “High-performance OC-12/OC-48 queue design prototype for input-buffered ATM switches,” in *INFOCOM’97*, (Kobe, Japan), pp. 20–28, Apr. 1997.