Dynamic Hardware Plugins (DHP): Exploiting Reconfigurable Hardware for High-Performance Programmable Routers

David E. Taylor, Jonathan S. Turner, John W. Lockwood
det3@arl.wustl.edu, jst@cs.wustl.edu, lockwood@arl.wustl.edu
Applied Research Laboratory
Washington University in Saint Louis
Campus Box 1045
One Brookings Drive
Saint Louis, MO 63130
USA
(314) 935-4845

***Abstract* -- This paper presents the Dynamic Hardware Plugins (DHP) architecture for implementing multiple networking applications in hardware at programmable routers. By enabling multiple applications to be dynamically loaded into a single hardware device, the DHP architecture provides a scalable mechanism for implementing high-performance programmable routers. The DHP architecture is presented within the context of a programmable router architecture which processes flows in both software and hardware. Possible implementations are described as well as the prototype testbed at Washington University in Saint Louis[1].**

***Keywords* -- Programmable router, reconfigurable hardware, active networking, port processor.**

I. INTRODUCTION

As researchers vigorously develop advanced control and processing schemes for programmable networks [1], there exists a need for a scalable router architecture capable of robust flow-specific processing at optical line speeds without prohibitively high per-port costs. As the quantity and diversity of streaming data and computationally intensive applications continues to increase, router architectures must respond with greater flexibility, processing capacity, and performance. With next-generation routers containing hundreds of ports, processing mechanisms must scale at a reasonable per-port cost.

Existing router architectures that provide sufficient flexibility and per-flow processing employ software processing environments containing multiple Reduced Instruction Set Computer (RISC) cores. Existing high-performance router architectures capable of data processing at optical line speeds employ Application Specific Integrated Circuits (ASICs) to perform parallel computations in hardware. However, these architectures often provide limited flexibility for deployment of new applications or protocols, and necessitate longer design cycles and higher costs than software-based solutions. Clearly, the an optimal router architecture must exhibit the flexibility available in software and the performance offered by hardware.

The diversity of networking applications and data flows suggests that dynamically reprogrammable processing environment is needed to cover the potential design space. While some applications performing limited processing at low data rates readily lend themselves to software implementation, a vast array of applications map well to hardware implementation due to high data rates, data regularities, and parallel operations. This implies that a viable solution to the programmable router problem should employ both software and reconfigurable hardware to process data flows. With the development of several multi-RISC core processor architectures and implementations, the problem of providing a scalable software processing environment is well investigated. The problem of adding a flexible and scalable hardware processing environment remains.

Traditionally used for low-volume prototyping and testing purposes, the reconfigurable hardware employed in Field Programmable Gate Arrays (FPGAs) provides a flexible hardware platform. Recently, reconfigurable hardware technology has made several compelling performance advances, identifying it as a possible solution for the programmable router node problem. New reconfigurable hardware devices tout approximately 1 million application logic gates, internal clock rates up to 200 MHz, over 100KB of on-chip memory, and partial-reconfiguration capability [2]. More impressive than the current technical statistics is the rate of progress due to

[1.] This research supported by NSF: ANI-0096052 and Xilinx, Inc.

architectural optimizations and silicon fabrication improvements: usable logic gate count increased by 10 times in two years; system clock frequency doubled in one year; I/O bandwidth quadrupled in two years; block and distributed on-chip memory capacity quadrupled in one year [3]. Reconfigurable hardware devices are clearly positioning themselves as viable options for flexible, high-performance systems.

The Dynamic Hardware Plugins (DHP) architecture employs reconfigurable hardware to provide a flexible hardware processing environment for programmable, multi-port routers. DHP allows multiple hardware applications, or plugins, to be dynamically loaded into a single device and run in parallel, providing a substantial amount of per-flow processing. With dedicated on-chip logic and memory resources provisioned for each plugin as well as arbitrated access to two types of off-chip memory resources, DHP supports a broad spectrum of applications. Results of several case studies of Advanced Encryption Standard (AES) implementations in software, FPGAs, and ASICs are used to show the potential performance and flexibility gains of the DHP architecture for networking applications in programmable routers.

II. Background and Related Work

Several schemes exist for delivering applications to a programmable router. Applications may be deployed at session setup via signalling protocols. Other schemes allow applications to be requested by incoming packets or carried by the packet for execution on the programmable router. With the exception of minor implementation details, the programmable router architecture discussion is orthogonal to application deployment mechanisms.

The router architecture presented in [5] provides a scalable software processing environment using elements with multiple RISC cores on a single device. This architecture readily lends itself to hardware processing integration and will be used as the departure point for discussing the DHP architecture.

Significant work has already been done in reconfigurable network hardware [4]. However, the previous approaches do not readily lend themselves to implementation in multi-port routers as the hardware requirements for a single flow of processing are prohibitively impractical. These previous approaches also do not provide ample memory resources to cover the design space of potential applications.

III. Programmable Router Architecture

Current routers capable of aggregate forwarding rates of terabits per second and link speeds of 2.4 Gb/s and 10 Gb/s set the standard for high-performance. Programmable routers need to achieve comparable performance to be considered a viable option for commercial applications. The router architecture described in [5] provides a scalable mechanism for processing data flows at router ports. The DHP architecture will be presented as an augmentation of this architecture to include a hardware processing environment.

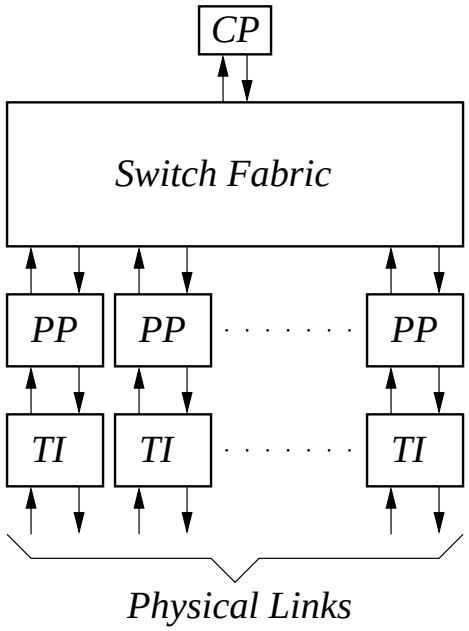

Figure 1: Programmable router architecture

As shown in Figure 1, the programmable router is built around a scalable multi-stage cell switching fabric as described in [6]. Based on this design, the Switch Fabric may be configured from ten to thousands of ports, each capable of supporting link rates of 2.4 Gb/s. Each physical link attaches to a Transmission Interface (TI) which converts data arriving on the link into a standard format for router input while performing the inverse operation on data destined for the output link. For fiber-optic links, this includes optoelectronic and serial/parallel signal conversion. Between the Transmission Interface and Switch Fabric is the Port Processor (PP). The Port Processor performs all of the flow classification, forwarding, queueing, and processing functions. The Port Processor architecture will be described in the next section. A Control Processor (CP) provides an external control interface and manages the Port Processors. The Control Processor is responsible for maintaining flow classification data structures and filters, as well as binding flows to applications at each Port Processor via flow identifiers. In larger systems, the CP may be a shared memory multiprocessor dimensioned to match the processing needs of the specific configuration.

IV. Port Processor Architecture

The Port Processor provides all of the necessary functionality to forward and process data flows as they pass through the router. The Port Processor architecture is detailed in Figure 2. The Packet Classification and Queueing (PCQ) element manages the flow of data through three device ports. The TI Port sends and receives data from the Transmission Interface, while the SW Port sends and receives data from the Switch Fabric. Data belonging to flows requiring processing are sent to and received from the processing elements via the Processing Element (PE) Port.

On the PCQ, the Packet Classifier performs a lookup operation on all packets arriving on the TI Port and attaches a flow identifier (flow ID) that identifies the destination of packets and type of processing, if any, that the packet is to receive at the Port Processor. The Packet Classifier maps each packet to a locally significant flow ID that is used to retrieve stored state information at other points in the system. It uses a general packet classification algorithm such as Pruned Tuple Space Search [7]. All data structures and flow IDs are maintained by the central Control Processor of the system.

After classification, packets are sent to the Queue Controller which manages output and application queues. Based on the flow ID, the Queue Controller places the packet on the appropriate queue. Packets not requiring processing are simply placed on the appropriate output queue. Packets requiring processing are placed on the queue associated with the application specified by the flow ID. The Queue Controller schedules packets from the set of application queues for output on the PE Port. Processed packets arriving on the PE Port are placed on the appropriate output queue.

A number of processing elements may reside in a chain at the PE Port of the PCQ. Figure 2 shows a Software Processing Element followed by a Hardware Processing Element. Note that the quantity and type of processing elements present at a Port Processor may be configured based on traffic demands at a particular port of the router. The Software Processing Element shown in Figure 2 consists of multiple RISC cores linked by a high speed I/O channel in a ring configuration. Processors are grouped in small clusters for the purpose of sharing access to off-chip memory interfaces. This architecture is a refinement of the architecture presented in [5], and is presented here mainly to set the context for the Dynamic Hardware Plugins architecture, which is the focus of this paper.

V. DHP Architecture

The Hardware Processing Element in Figure 2 utilizes the Dynamic Hardware Plugins (DHP) architecture to add flexible hardware processing capability to the Port Processor. DHP employs reconfigurable hardware to allow multiple applications to be dynamically loaded into hardware plugins and run in parallel on a single device. Data flows may pass through permutations of hardware plugins, allowing for substantial amounts of per-flow processing. In order to support a broad spectrum of applications, each plugin possesses dedicated on-chip logic and memory resources as well as access to two types of arbitrated off-chip memory resources.

In order to facilitate the current architectural discussion and a later discussion of implementation options, the DHP architecture is divided into two major parts: hardware plugins and infrastructure. Hardware plugins are the hardware components that may be dynamically reconfigured to support new applications. Infrastructure consists of the static control and datapath components of the DHP architecture. The infrastructure components collectively route packets to plugins and I/O ports, dynamically reconfigure hardware plugins, interface to external memory devices and arbitrate access among the contending pool of applications. The following subsections discuss the major divisions of the DHP architecture and their associated components in detail.

A. Infrastructure

The infrastructure, denoted by the shaded blocks in Figure 2, is the required collection of static control and datapath components to support dynamic, modular hardware applications. The infrastructure provides common services to hardware plugins and hides details of memory device timing. By providing a standard interface for plugins, the infrastructure provides the equivalent of an API to allow hardware developers to more easily design modular applications that work together.

1) Data I/O and Flow Control

As shown in Figure 2, the DHP architecture arranges hardware plugins in a slotted ring with each ring interface labelled as an Input Output Controller (IOC). A ring architecture was chosen in preference to a bus because rings can be operated at higher clock frequencies than buses due to their simple point-to-point connections and the resulting reduction in capacitive loading. The ring is better in this context than a crossbar since it allows a single plugin to make use of the full ring bandwidth if necessary. A crossbar capable of providing similar bandwidth to each plugin requires substantially more processing resources. While rings do add latency to data transfers, a suitable hardware implementation can keep these latencies to well under a microsecond in typical configurations. In order to keep up with a link rate of

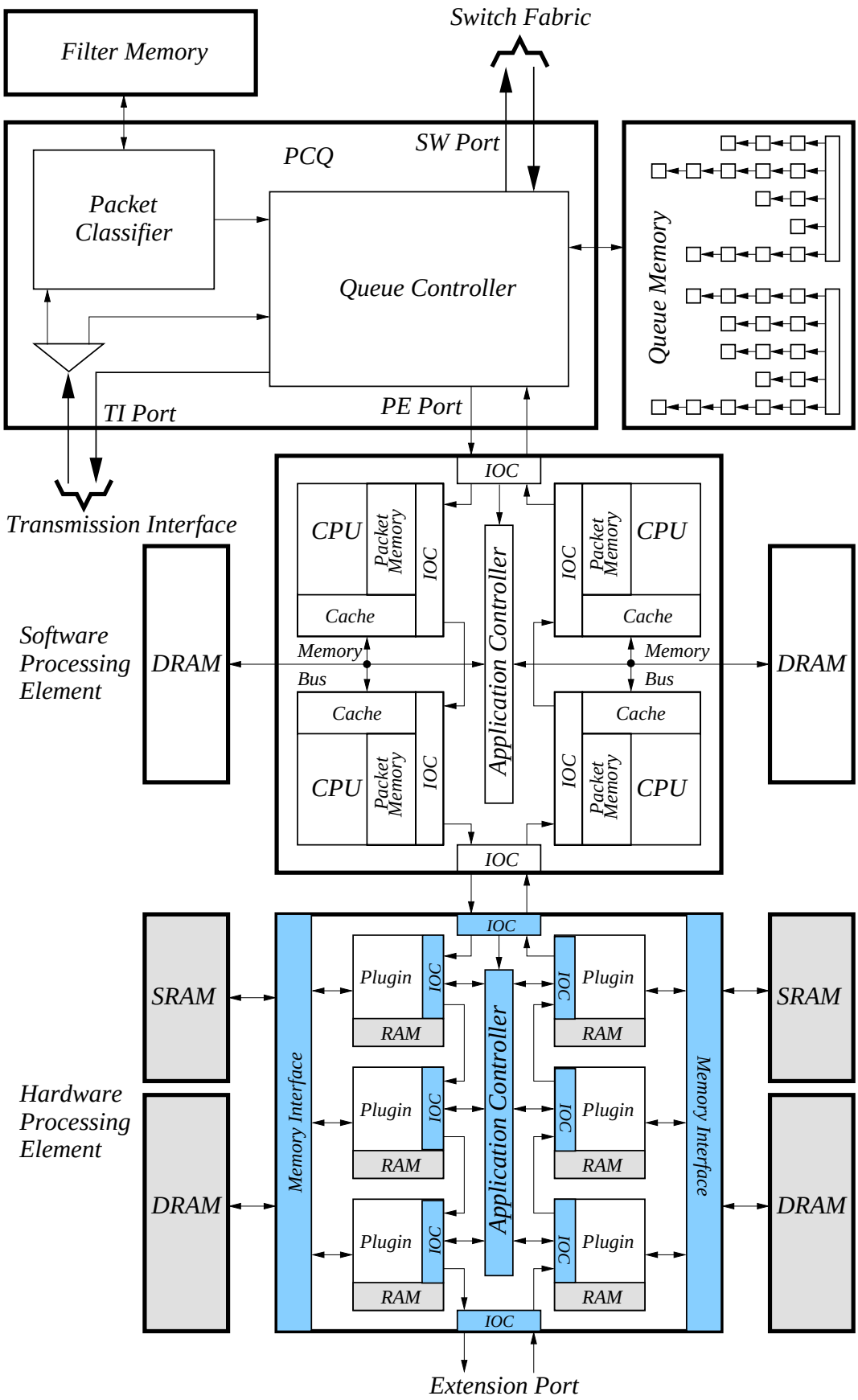

Figure 2: Port Processor (PP) architecture with Hardware and Software Processing elements. The Hardware Processing Element employs the Dynamic Hardware Plugins (DHP) architecture.

2.4Gb/s, the ring must have a minimum bandwidth of 4.8 Gb/s to allow hardware plugins to process both ingress and egress data flows at the link rate. A 32-bit wide ring operating at 200 MHz provides a raw bandwidth of 6.4 Gb/s, providing sufficient extra bandwidth to handle internal overheads and keep contention low.

Note that an IOC is provided for each hardware plugin while two IOCs interface to upstream and downstream elements. The upstream IOC may interface to another processing element or directly to the PCQ. The downstream IOC interfaces only to other processing elements. The ring protocol transfers fixed size units with a busy/idle bit in the first word of each transmission slot. The first word also includes a flow control bit vector with one bit for each IOC on the chip. An IOC sets its bit to signal congestion. A second bit vector is used to enable fair access to the ring. Each plugin with data queued for transmission on the ring sets its bit and paces its transmissions on the ring based on the number of bits set by other plugins. Additional fields in this word identify a ring and slot number of the destination application for the packet. The ring number identifies a unique processing element in the chain, while the slot number specifies the hardware plugin containing the destination application. For packets requiring processing by more than one application, the third bit vector is modified to address the next application. Upon completion of a packet, applications identify the correct ring and slot number of subsequent applications via locally available state information.

As shown in Figure 2 the upstream IOC contains an additional port to the Application Controller. When new applications are to be loaded into the hardware plugins, the upstream IOC must pass control messages and application data to the Application Controller. While a hardware plugin undergoes reconfiguration, the associated IOC passes data to the next IOC in the ring. This mechanism allows applications to be dynamically loaded into hardware plugins without interrupting the flow of data through the processing ring.

2) Application Controller

The Application Controller manages the dynamic reconfiguration of hardware plugins to support new applications. Hardware applications arrive as bitfiles to the Application Controller. Bitfiles specify the logic operations, signal routing, and on-chip memory configuration for the hardware application. As bitfiles may be loaded from local memory or remotely over the network, the Application Controller must assemble, buffer, and ensure the correctness of the bitfile prior to loading it into the hardware plugin. Bitfile integrity can be maintained via checksums and reliable transport protocols.

Prior to reconfiguring the hardware plugin, the Application Controller initiates a handshake with the application to prevent data and flow state loss. If the application is not idle, it must stop accepting packets and finish processing current packets. Applications may define appropriate breakpoints for reconfiguration based on the type of flow processing it is performing. Control messages may be sent from applications to the PCQ to halt packet forwarding at breakpoints. For deployment of application revisions, applications may copy flow state to off-chip memory for the new revision to use once it has been loaded into the hardware plugin. Once the application has ensured that no data or relevent flow state will be lost, it returns a handshake to the Application Controller. At this point, the IOC routes all arriving packets to the next IOC in the ring. The Application Controller then loads the new application into the hardware plugin by writing the application bitfile to the reconfigurable logic.

The amount of time required for plugin reconfiguration depends on the size of the plugin and the complexity of the application. Current FPGA technologies do not place a strong emphasis on high reconfiguration speeds. However, as discussed in a later section the time required to configure a current generation FPGA with a complex application such as an encryption cipher requires on the order of 5 ms. While this time is not so long as to make DHP impractical with current technology, the current programming rates of 66 MB of configuration data per second must increase for next generation technology to be suitable for use in programmable routers. As designers continue to develop systems that demand high-speed device configuration [8], it is likely that FPGA vendors will need to respond with faster reconfiguration mechanisms.

Once all configuration data is loaded into the hardware plugin, the Application Controller initiates a localized reset to the hardware plugin. The Application Controller waits for a handshake from the application. Once the application is initialized and ready, it completes the handshake with the Application Controller. The Application Controller responds with a control message to the CP, which updates the descriptor table used by the Packet Classifier. The IOC then routes packets with matching descriptors to the application.

3) Memory Interfaces

In order to cover the design space of potential hardware applications, DHP provides access to two types of off-chip memory resources. Banks of Synchronous Random Access Memory (SRAM) provide storage for per flow state and computations requiring low-latency accesses, while banks of Dynamic Random Access Memory

(DRAM) provide ample resources for memory intensive applications. The Memory Interfaces shown in Figure 2 arbitrate access among the hardware plugins while insulating applications from device-specific timing specifications.

The type of hardware technology used to implement the hardware processing element limits the number of pins available for interfacing to off-chip memory devices. Current devices are capable of supporting two SRAM devices and two DRAM devices; therefore, this configuration will be used for the purpose of this discussion. Due to the wide array of memory devices and technologies available, the type of SRAM and DRAM devices employed in a particular system will likely be a function of size, speed, and cost constraints. Systems running applications that require high-bandwidth access to large amounts of memory may employ DRAM technologies, such as Rambus, to meet performance constraints [9]. Implementation of such complex memory interfaces requires more on-chip hardware resources and will be discussed in the Implementation section. Other system designers may wish to reduce cost by using Synchronous Dynamic Random Access Memory (SDRAM) devices.

To allow for flexibility in selecting external memory devices, the Memory Interfaces provide a standard interface to hardware plugins abstracting them from device-specific timing and control signalling. The Memory Interfaces provide each plugin with independent access to both memory types, hence applications are free to utilize both types of off-chip memory resources in parallel. Hardware plugins gain access to off-chip memory via a simple grant/request handshake. The Memory Interface services requests in a round-robin fashion. Once access is granted, applications may issue read, write, burst read, and burst write commands. Starvation avoidance is achieved by plugins monitoring the status of the grant/request signals. When other plugins contend, the plugin currently accessing memory must release memory at the conclusion of the current transaction.

B. Hardware Plugins

Hardware plugins provide applications with the reconfigurable logic and memory resources to process data flows. In this context, hardware plugins are the physical hardware structures that may be configured to implement various networking applications. The reconfigurable logic resources include logic gates, lookup tables, flip-flops, multiplexors, demultiplexors, and signal routing matrices. On-chip Random Access Memory (RAM) may be configured to implement queues and multi-port memories.

1) Interface

In order to design modular applications for use in the DHP, a standardized hardware plugin interface is necessary. Like an API for software, hardware plugins must interface to a static set of ports for data I/O, control, and external memory. As shown in Figure 3, the hardware plugin interface includes off-chip SRAM and DRAM interfaces, IOC interface, and Application Controller interface. Each application may also define its own interface to on-chip RAM.

The interface to off-chip DRAM includes grant and request signals for the arbitration handshake, memory command signals, address lines, and tri-state data lines. Similarly, the off-chip SRAM interface includes grant and request signals, memory command signals, and address lines. However, this interface employs separate input and output data lines to allow for pipelined memory reads and writes. For low-latency state and data storage, applications may define unique interfaces to on-chip RAM. Reconfigurable hardware technology allows these resources to function as multi-port memories, queues, and large register files.

The IOC interface includes input and output queue interfaces. The input queue interface employs a “not empty” status flag, while the output interface uses a “full” status flag. To keep the design uniform, the queue data paths are the same width as the ring. The Application Controller interface provides applications with a system clock, local reset, and enable/ready signals for the reconfiguration handshake with the Application Controller. Applications may use subdivided or multiplied versions of the system clock to suit design needs.

2) Applications

While the focus of this paper is not a performance comparison of hardware and software applications, it is important to identify the types of applications that benefit from the DHP architecture. Any computationally intensive application operating on streaming data at high rates is a likely candidate. Potential applications also need to contain operations that may be performed in parallel or pipelined. Purely sequential computations cannot take full advantage of the inherent benefits of hardware implementations.

One of the most widely used applications which is also crucial to the growth of the Internet as a commercial tool is encryption. Since every byte must be manipulated in order to properly encrypt a data block, encryption is a computationally expensive application. Due to the nature of the computations performed, encryption is highly amenable to hardware implementation. Results of case studies of the new Advanced Encryption Standard (AES)

will be used to illustrate the potential performance of DHP for networking applications.

In order to select an algorithm for AES, the National Institute of Standards and Technology (NIST) and several independent research groups analyzed the security and performance of the finalist algorithms for both software and hardware implementations [10][11][12]. Based on these analyses, Rijndael was selected as the algorithm for AES [13]. In order to provide a baseline performance comparison of software and FPGAs, the authors of [11] implemented and analyzed an iterative version of the Rijndael algorithm that provided encryption, decryption, and key-scheduling for 128-bit keys operating over 10 rounds on 128-bit data blocks in a Xilinx FPGA. This implementation achieved a throughput of 353 Mb/s, providing a factor of 11.15 speedup over comparable software implementations that achieved 31.64 Mb/s. This implementation would occupy approximately 20% of the available resources of the largest current generation FPGA, a Xilinx Virtex 3200E, and would require on the order of 5 ms for device configuration. While these results show significant performance gains, NIST cited case studies of ASIC implementations of the Rijndael algorithm achieving throughputs of 5.16 Gb/s, a factor of 163 speedup over software [10]. This level of performance was achieved through fully pipelined architectures as opposed to the iterative architectures used in [11]. Fully pipelined architectures require significantly more resources, making them impractical for use in current generation FPGAs and the DHP architecture. However, for implementation in programmable routers a higher performance FPGA implementation of AES is required.

The authors of [12] implemented several architectural variants of the Rijndael algorithm in an FPGA. Their analysis focused solely on encryption throughput, operating under the assumption that key-scheduling delays can be masked by a suitable parallel implementation. This analysis is relevant to the programmable router discussion, as throughput is the metric of interest and it is likely that encryption and decryption will occur in separate hardware plugins. In this analysis, the authors found that a 5-stage partial-pipeline with a single-stage sub-pipeline architecture of Rijndael algorithm achieved a throughput of 1.94 Gb/s. While this implementation required nearly twice the amount of device resources as the iterative implementation in the aforementioned study, it occupies less than 40% of the largest current generation FPGA.

Based on these results, a single hardware plugin could encrypt 80% of the traffic carried on an OC-48 link. Achieving this level of performance in software would require distributing the computation over 60 RISC cores, an exorbitant amount of resources for a single application operating on a single link. These results clearly strengthen the case for employing reconfigurable hardware in reprogrammable routers. Unlike ASIC implementations, the DHP architecture allows for new encryption standards such as AES to be deployed in a matter of milliseconds.

New streaming data services such as audio and video bridging for video conferencing also provide ideal hard-

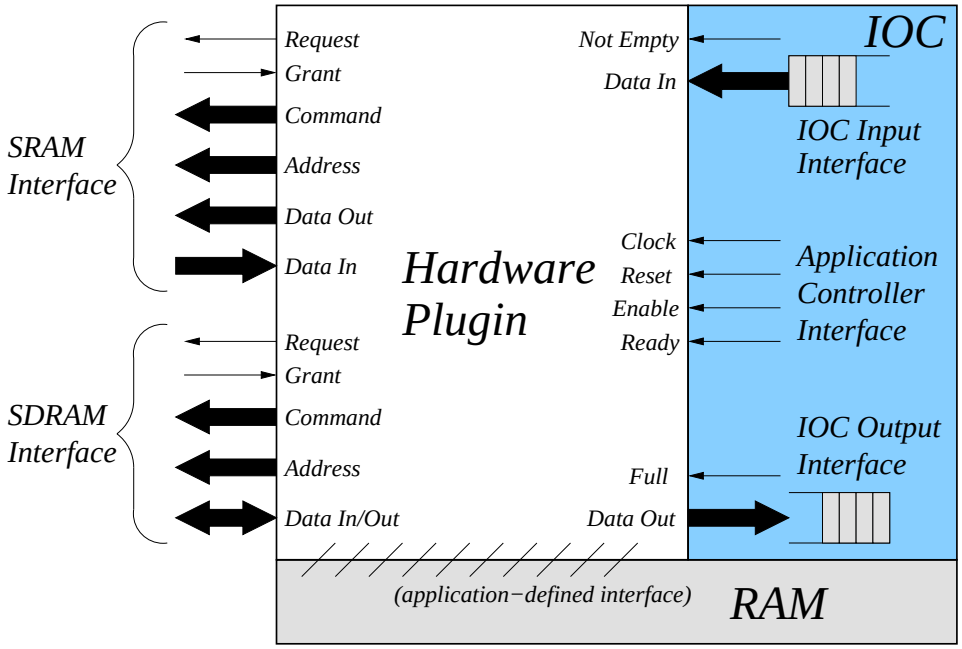

Figure 3: Hardware Plugin interface with static interfaces to infrastructure components. Applications define interfaces and configurations for on-chip memory.

ware plugin applications. Multi-service routing and multicast support are also ideal candidates for hardware implementation. With the proliferation of Hardware Description Languages (HDLs) as common tools for designing hardware applications, many applications currently implemented in ASICs can be easily ported for DHP implementation.

VI. Implementation

Due to strides in current FPGA technology, the Dynamic Hardware Plugins architecture can be implemented in a single FPGA. As device speeds and densities continue to increase, the quantity and performance capabilities of hardware plugins available on a single device will likewise increase. Providing dynamic, modular plugins surrounded by static control structures in a single device physically translates to partially reprogramming a running FPGA at the port of a router. This is a non-trivial task that is the focus of ongoing research at Washington University in Saint Louis. A significant part of the solution involves new CAD tools capable of targeting specific regions of a device, producing partial reprogramming bitfiles, reserving logic and routing resources, and locking signals for static plugin interfaces. While many of these capabilities exist in one form or another within current CAD tool suites, execution of this task requires an enormous amount of effort. While this is a significant area of research, its discussion is beyond the scope of this paper.

While an FPGA is a readily available device for implementing the DHP architecture, it also leaves room for improvement. One refinement would be to use static logic for infrastructure components, enabling higher performance with more efficient resource usage. While FPGAs support several I/O standards, the types of memory devices available to system designers are limited by those supported by FPGA vendors. Adding high performance memory interfaces, such as Rambus, would provide substantially better performance.

An intriguing implementation option for the DHP architecture is a mixed ASIC/FPGA. By hand-crafting the IOC ring, Application Controller, and Memory Interfaces in ASIC technology, greater I/O performance could be achieved for off-chip data transfers and memory transactions as well as faster plugin configuration. Given the same die size, this would also result in a more area efficient infrastructure implementation providing more area for reconfigurable hardware; hence, more logic and memory resources per hardware plugin or more plugin slots per device. However, unlike FPGA implementations the mixed ASIC/FPGA implementation does not provide for plugin size dimensioning based on projected application demands.

VII. Prototype Testbed

In order to prototype the Dynamic Hardware Plugins architecture operating in a Port Processor of a multi-port programmable router, several research systems designed and built at Washington University in Saint Louis are used in combination [14]. The WUGS 20, an 8 port ATM switch providing 20 Gb/s of aggregate throughput, is used for the Switch Fabric. This switching core is based upon a multi-stage Benes topology, supports up to 2.4 Gb/s link rates, and scales up to 4096 ports for an aggregate throughput of 9.8 Tb/s. The Smart Port Card (SPC) is used to prototype the software processing element [15]. It employs an embedded microprocessor, memory, and custom network interface ASIC to process network data flows. The Field Programmable Port Extender (FPX) is used to prototype the Dynamic Hardware Plugins architecture [16][17]. It employs two FPGAs, one acting as the Network Interface Device (NID) and the other as the Reprogrammable Application Device (RAD). The RAD FPGA has access to two 1 MB Zero-Byte Turnaround (ZBT) SRAMs and two 64MB SDRAM modules. A diagram of the FPX is shown in Figure 4. Both the SPC and FPX are implemented on Printed Circuit Boards (PCBs) of the same form factor as the WUGS transmission interfaces. Hence, each port of the WUGS may be fitted with different FPX/SPC combinations. Photographs of the FPX and the FPX in the WUGS are shown in Figure 5.

VIII. Conclusion

Dynamic Hardware Plugins provides a scalable mechanism for building high-performance, multi-port routers capable of robust per flow processing. As reconfigurable hardware technology continues to offer higher performance via denser logic and memory resources at faster clock rates, the amount and diversity of per flow processing made available by the DHP architecture likewise increases. Implementing networking applications in hardware provides performance levels either not achievable in software, or achievable only with significantly more hardware resources and complex control mechanisms. By allowing multiple hardware applications to be dynamically loaded into a single device, the DHP architecture is a flexible, parallel, hardware processing mechanism. As applications are developed, the prototype testbed at Washington University in Saint Louis provides an ideal platform for performance analysis and further research into reconfigurable network hardware.

Acknowledgment

The authors would like to thank David Parlour and Xil-

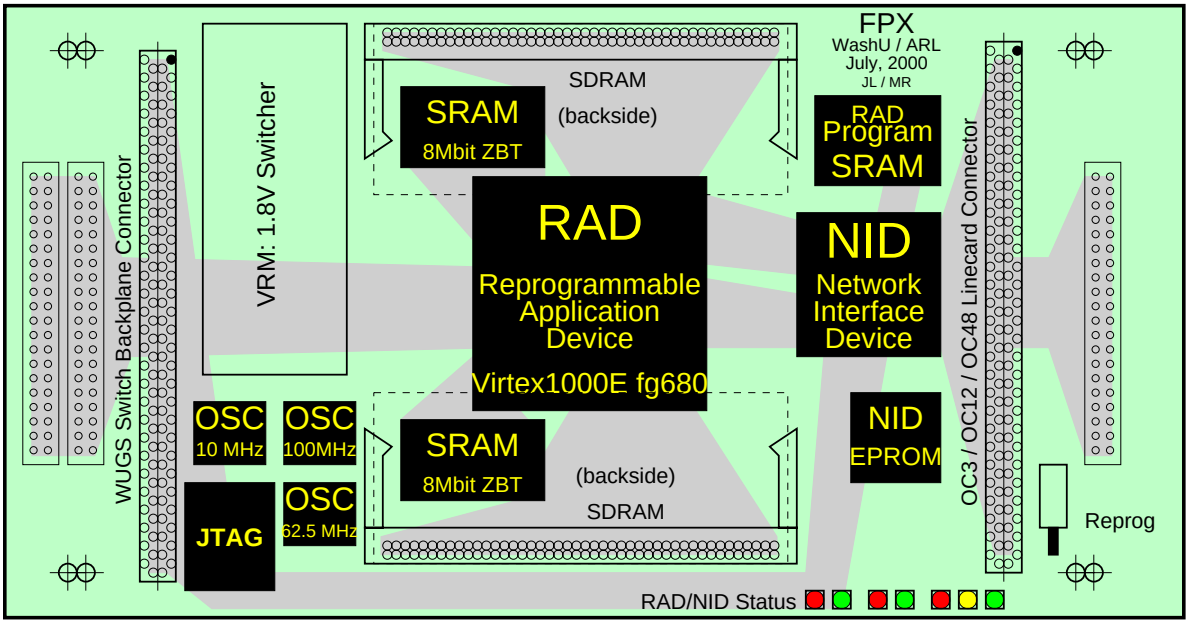

Figure 4: Diagram of the Field Programmable port eXtender (FPX) used to prototype the Dynamic Hardware Plugins (DHP) architecture.

inx, Inc. for their support and efforts to aid in this research. The authors would also like to thank Naji Naufel for his contributions to the FPX project.

REFERENCES

[1] Tennenhouse, D.L., Smith, J.M., Sincoskie, W.D., Wetheral, D.J., Minden, G.J. "A Survey of Active Network Research", *IEEE Communications*, 35, 1, January 1997, 80-86.

[2] Xilinx, Inc. "Virtex-E 1.8V Field Programmable Gate Arrays", Advance Product Specification, February 29, 2000, San Jose, CA.

[3] Xilinx, Inc. "Xilinx Unveils New FPGA Architecture to Enable High-Performance, Ten Million System-Gate Designs" Press Release, May 22, 2000; San Jose, CA

[4] Hadzic, I., Smith, J. "On-the-fly Programmable Hardware for Networks", Proceedings of GLOBECOM 1998.

[5] Wolf, T., Turner, J.: "Design Issues for High Performance Active Routers," Proceedings of International Zurich Seminar on Broadband Communications, Zurich, Switzerland, February 2000.

[6] T. Chaney, A. Fingerhut, M. Flucke, J. Turner, "Design of a Gigabit ATM Switch", Proc. of INFOCOM 97, Kobe, Japan.

[7] Srinivasan, V., Suri, S., Varghese, G., "Packet Classification using Tuple Space Search", Proc. of SIGCOMM 99, Cambridge, Mass.

[8] H. Schmit, "Incremental Reconfiguration for Pipelined Applications," Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines, pp. 47-55, 1997.

[9] Warmke, R., "Designing a Multimedia Subsystem with Rambus DRAMs", Multimedia Systems Design, March 1998, Miller Freeman, Inc.

[10] J. Nechvatal, E. Barker, L. Bassham, W. Burr, M. Dworkin, J. Foti, E. Roback, ''Report on the Development of the Advanced Encryption Standard (AES)'', Computer Security Division Information Technology Laboratory, National Institute of Standards and Technology, Technology Administration, U.S. Department of Commerce, October 2, 2000.

[11] A.Dandalis, V.K. Prasanna, J.D.P. Rolim, "A Comparative Study of Performance of AES Final Candidates Using FPGAs", AES3: The Third Advanced Encryption Standard (AES) Candidate Conference, Gaithersburg, MD, April 2000.

[12] A. Elbirt, et al., ''An FPGA Implementation and Performance Evaluation of the AES Block Cipher Candidate Algorithm Finalists,'' AES3: The Third Advanced Encryption Standard (AES) Candidate Conference, National Institute of Standards and Technology, Gaithersburg, MD, April 2000.

[13] J. Daemen, V. Rijmen, ''AES Proposal: Rijndael,'' First Advanced Encryption Standard (AES) Conference, (Ventura, California, USA), 1998.

[14] Turner, J., Choi, S., Decasper, D., DeHart, J., Keller, R., Lockwood, J., Wolf, T., "Design of a Flexible Open Platform for High Performance Active Networks", Proceedings of the Allerton Conference, 10/99.

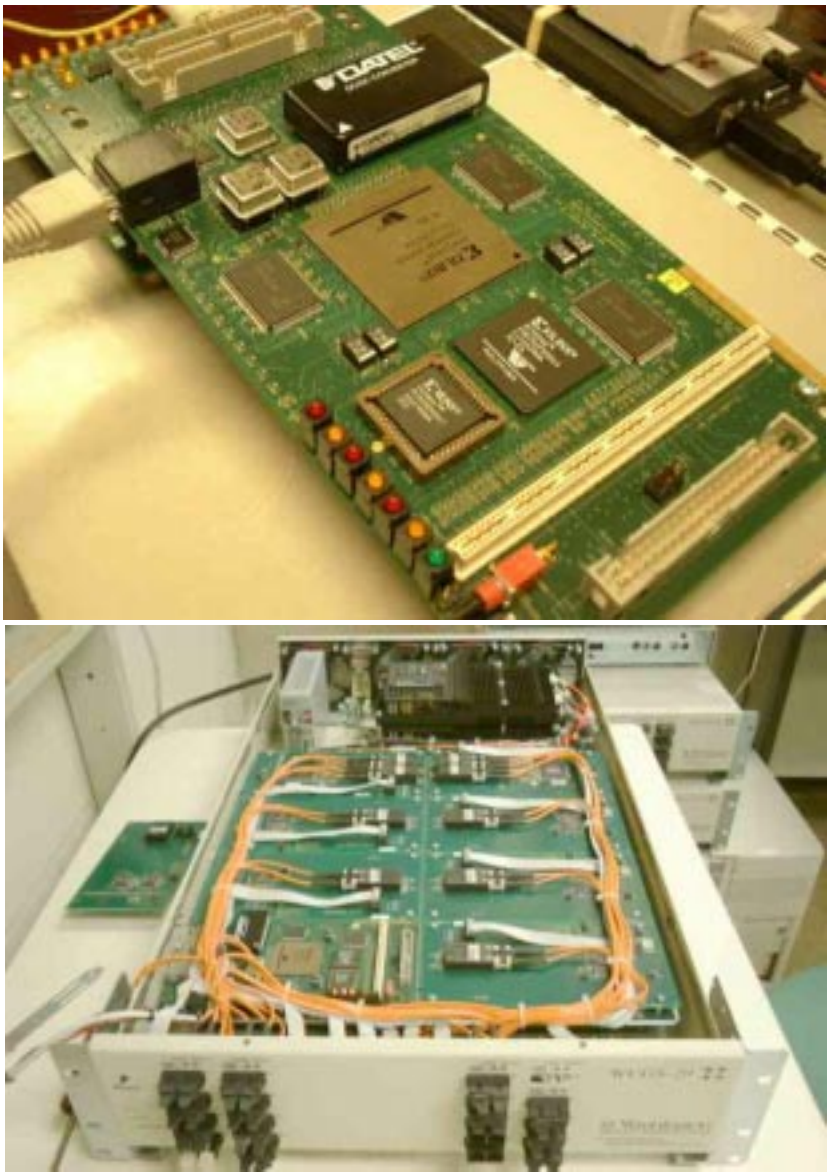

Figure 5: Prototype environment for the DHP architecture. a.) Photograph of the Field Programmable port eXtender (FPX). b.) Photograph of an FPX in a WUGS with the line card removed for visibility.

[15] Decasper, D., Parulkar, G., Choi, S., DeHart, J., Wolf, T., Plattner, B.: "A Scalable High-Performance Active Network Node". IEEE Network, Vol. 13, No. 1, January/February 1999.

[16] Lockwood, J., Turner, J., Taylor, D. “Field Programmable Port Extender (FPX) for Distributed Routing and Queuing”, FPGA 2000: Eighth ACM International Symposium on Field-Programmable Gate Arrays, Monterey, CA, 2/00.

[17] Lockwood, J., Naufel, N., Taylor, D., Turner, J., “Reprogrammable Network Packet Processing on the Field Programmable Port Extender (FPX)”, FPGA 2001: Ninth ACM International Symposium on Field-Programmable Gate Arrays, Monterey, CA, 2/01.