2001-20

Implementation of an Open Multi-Service Router

Authors: <a href="http://www.arl.wustl.edu/~fredk/">Kuhns, F.,</a><a href="http://www.arl.wustl.edu/~jdd/">

Type of Report: Other

Implementation of an Open Multi-Service Router

Fred Kuhns†, John DeHart†, Ralph Keller†, John Lockwood†, Prashanth Pappu†,
Jyoti Parwatikar†, Ed Spitznagel†, David Richards‡, David Taylor‡, Jon Turner† and Ken Wong†
{fredk,jdd,keller, lockwood,prashant,jp,ews1,wdr,det3,jst,kenw}@arl.wustl.edu
†Department of Computer Science and the Applied Research Laboratory
‡Department of Electrical Engineering and the Applied Research Laboratory
Washington University, St. Louis, MO 63130, USA

1. Introduction

In the last decade, the Internet has undergone a fundamental transformation, from a small-scale network serving academics and select technology companies, to a global infrastructure serving people in all walks of life and all parts of the world. As the Internet has grown, it has become more complex, making it difficult for researchers and engineers to understand its behavior and that of its many interacting components. This increases the challenges faced by those seeking to create new protocols and technologies that can potentially improve the Internet's reliability, functionality and performance. At the same time, the growing importance of the Internet is dramatically raising the stakes. Even small improvements can have a big payoff.

In this context, experimental studies aimed at understanding how Internet routers perform in realistic network settings, are essential to any serious research effort in Internet technology development. Currently, academic researchers have two main alternatives for experimental research in commercial routers and routing software. With commercial routers, researchers are generally limited to treating the router as a black box with the only access provided by highly constrained management interfaces. The internal design is largely hidden, and not subject to experimental modification.

The other alternative for academic researchers is to use routing software, running on standard computers. Open source operating systems, such as Linux and NetBSD have made this a popular choice. This alternative has the advantage that it provides direct access to all of the system's functionality and provides complete extensibility. The growing performance demands of the Internet have made the internal design of high performance routers far more complex. Routers now support large numbers of gigabit links and use dedicated hardware to implement many protocol processing functions. Functionality is distributed among the line cards that interface to the links, the control processors that provide high level management and the interconnection network that moves packets from inputs to outputs. The highest performance systems use multistage interconnection networks capable of supporting hundreds or even thousands of 10 Gb/s links. To understand how such systems perform, one must work with systems that have the same architectural characteristics. A single processor with a handful of relatively low speed interfaces, uses an architecture which is both quantitatively and qualitatively very different. The kinds of issues one faces in systems of this sort are very different from the kinds of issues faced by designers of modern high performance routers. If academic research is to be relevant to the design of such systems, it needs to be supported by systems research using comparable experimental platforms.

The Multi-Service Router (MSR) being developed at Washington University provides an ideal platform for advanced networking research in the increasingly complex environment facing researchers and technology developers. It is built around a switch fabric that can be scaled up to large numbers of ports. While typical research systems have small port counts, they do use the same parallel architecture used by much larger systems, requiring researchers to address in a realistic way many of the issues that arise in larger systems. The MSR has embedded, programmable processors at every link interface, allowing packet processing at these interfaces to be completely flexible. An extension to the MSR architecture, which is now in progress, will enable all packet processing to be implemented in hardware, allowing wire-speed forwarding at gigabit rates. The design of all software and hardware used in the MSR is being placed in the public domain, allowing it to be studied, modified and reused by researchers and developers interested in advancing the development of open, extensible, high performance Internet routers.

Section 2 describes the overall system architecture and some novel hardware components. Section 3 describes the design and implementation of system-level processing elements and some of the design issues related to its distributed architecture. Section 4 describes processing done at the port processors. Section 5 describes performance measurements of our early prototype which uses a software implementation of our packet forwarding engine and active packet processor. The measurements quantify the systems ability to forward packets and provide fair link access. Finally, Section 6 closes with final remarks on the current status of the system and future extensions.

2. System Overview

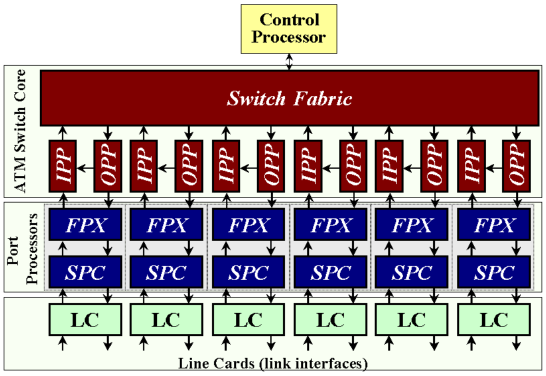

Figure 1: MSR Hardware Configuration

The Washington University MSR is designed to be a scalable, high-performance, open platform for conducting network research. It employs highly reconfigurable technology (programmable hardware and dynamic software modules) to provide high-speed processing of both IP packets (with and without active processing) and ATM cells. Figure 1 shows the overall architecture of the MSR and its main components: Control Processor (CP), ATM switch core, Field Programmable port eXtenders (FPXs), Smart PortCards (SPCs) and Line Cards (LCs).

The main function of the router is to forward packets at a high speed from its input side to its output side. The system uses a multistage interconnection network with dynamic routing and a small internal speed advantage (i.e., the internal data paths can forward packets at a faster rate than the external links) to connect the input side Port Processors (PPs) to the output side PPs. A PP can be either a Field Programmable port eXtender (FPX) and/or a Smart PortCard (SPC). An FPX is a reprogrammable hardware device, and an SPC is a general-purpose processor. These PPs perform packet classification, route lookup and packet scheduling.

The system employs a number of interesting techniques aimed at achieving high performance and flexibility. A *distributed queueing* algorithm is used to gain high throughput even under extreme overload. The PPs use a packet classification algorithm that can run at wire speed when implemented in hardware. The CP runs open source route daemons that support standard protocols such as OSPF as well as the MSR's own flow-specific routing protocol. Furthermore, the key router functions are efficiently distributed among its hardware components by exploiting the high bandwidth and connection-oriented circuits provided by the ATM switch core. The remainder of this section gives an overview of the MSR hardware components.

2.1. Control Processor

The Control Processor (CP) runs software that directly or indirectly controls and monitors router functions such as port status, resource usage and packet classification tables used in the Port Processors (PPs). Some of this processing is described in Sections 3 and 4. The CP is connected to one of the MSR's ports and uses ATM control cells to control and monitor PP activity.

2.2. Switch Fabric and Line Cards

The MSR's ATM switch core is a Washington University Gigabit ATM Switch (WUGS)[1, 2]. The current WUGS has eight (8) ports with Line Cards (LCs) capable of operating at rates up to 2.4 Gb/s and supports ATM multicasting using a novel cell recycling architecture.

Each LC provides conversion and encoding functions required for the target physical layer device. For example, an ATM switch link adapter provides parallel-to-serial, encoding, and optical-to-electrical conversions necessary for data transmission over fiber using one of the optical transmission standards, e.g., SONET. Current LCs include a dual 155 Mb/s OC-3 SONET [3] link adapter, a 622 Mb/s OC-12 SONET link adapter, a 1.2 Gb/s Hewlett Packard (HP) G-Link [4] link adapter, and a dual 1.2 Gb/s HP G-Link adapter. A gigabit ethernet LC is currently being designed.

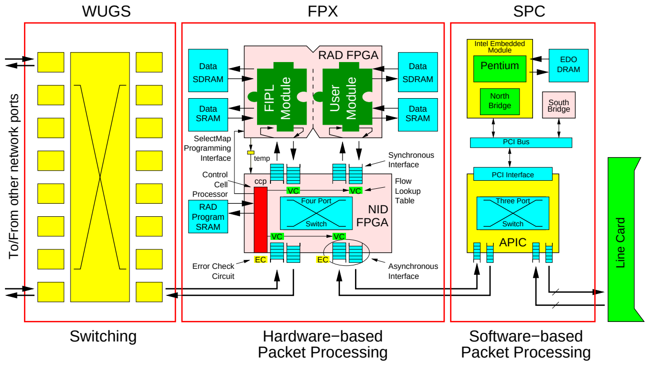

Figure 2: An FPX/SPC Port Processor

2.3. Port Processors

Commercial switches and routers already employ complex queueing and packet filtering mechanisms. However, this is usually accomplished through specialized integrated circuits. Figure 2 shows how the MSR uses PPs made up of a general-purpose processor (the SPC) with a reprogrammable hardware device (the FPX) to provide these mechanisms. This implementation approach takes advantage of the benefits of a cooperative hardware/software combination [5, 6]. Although the SPC is capable of performing all port functions, a high-speed configuration uses both the FPX and SPC. The FPX acts as a forwarding engine [7], and the SPC acts as a network processor handling non-standard processing (e.g., active packet, IP options).

Field Programmable Port Extender (FPX): The FPX is a programmable hardware device that processes packets as they pass between the WUGS backplane and the line card (shown in the middle of Figure 2). All of the logic on the FPX is implemented with two FPGA devices: the Network Interface Device (NID) and the Reprogrammable Application Device (RAD) [7]. The FPX is implemented on a 20 cm × 10.5 cm printed circuit board that interconnects the FPGAs with multiple banks of memory.

The Network Interface Device (NID) controls how packets are routed to and from its modules. It also provides mechanisms to load hardware modules over the network. These two features allow the NID to dynamically load and unload modules on the RAD without affecting the switching of other traffic flows or the processing of packets by the other modules in the system [8].

As shown in the lower-center of Figure 2, the NID has several components, all of which are implemented on a Xilinx Virtex XCV-600E FPGA device. It contains: 1) A four-port switch to transfer data between ports; 2) Flow look-up tables on each port to selectively route flows; 3) An on-chip *Control Cell Processor* to process control cells that are transmitted and received over the network; 4) Logic to reprogram the FPGA hardware on the RAD; and 5) Synchronous and asynchronous interfaces to the four network ports that surround the NID.

A key feature of the FPX is that it allows the MSR to perform packet processing functions in modular hardware components. As shown in the upper-center of Figure 2, these modules are implemented as regions of FPGA logic on the RAD. A standard interface has been developed that allows a module to process the streaming data in the packets as they flow through the module and to interface with off-chip memory [9]. Each module on the RAD connects to one Static Random Access Memory (SRAM) and to one wide Synchronous Dynamic RAM (SDRAM). In total, the modules implemented on the RAD have full control over four independent banks of memory. The SRAM is used for applications that need to implement table lookup operations such as the routing table for the Fast IP Lookup (FIPL) module. The other modules in the system can be programmed over the network to implement user-defined functionality [10].

Smart Port Card (SPC): As shown in Fig. 3, the Smart Port Card (SPC) consists of an embedded Intel processor module, 64 MBytes of DRAM, an FPGA that provides south bridge functionality, and a Washington University APIC ATM host-network interface [11]. The SPC runs a version of the NetBSD operating system [12] that has been substantially modified to support fast packet forwarding, active network processing and network management.

The Intel embedded module contains a 166 MHz Pentium MMX processor, north bridge [13] and L2 cache. The “System FPGA” provides the functionality of the south bridge chip [14] found in

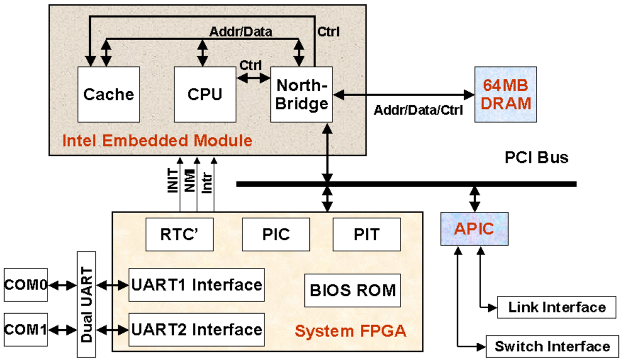

Figure 3: Block Diagram of the Smart Port Card (SPC)

a normal Pentium system and is implemented using a Xilinx XC4020XLA-08 Field Programmable Gate Array (FPGA) [15]. It contains a small boot ROM, a Programmable Interval Timer (PIT), a Programmable Interrupt Controller (PIC), a dual UART interface, and a modified Real Time Clock (RTC'). See [16] for additional details.

On the SPC, ATM cells are handled by the APIC [17, 18]. Each of the ATM ports of the APIC can be independently operated at full duplex rates ranging from 155 Mb/s to 1.2 Gb/s. The APIC supports AAL-5 and is capable of performing segmentation and reassembly at the maximum bus rate (1.05 Gb/s peak for PCI-32). The APIC directly transfers ATM frames to and from host memory and can be programmed so that cells of selected channels pass directly from one ATM port to another.

We have customized NetBSD to use a disk image stored in main memory, a serial console, a self configuring APIC device driver and a "fake" BIOS. The fake BIOS program acts like a boot loader: it performs some of the actions which are normally done by a Pentium BIOS and the NetBSD boot loader during power-up.

The above hardware provide the foundation for implementing the system functionality to be described in the following two sections.

3. System-Level Processing

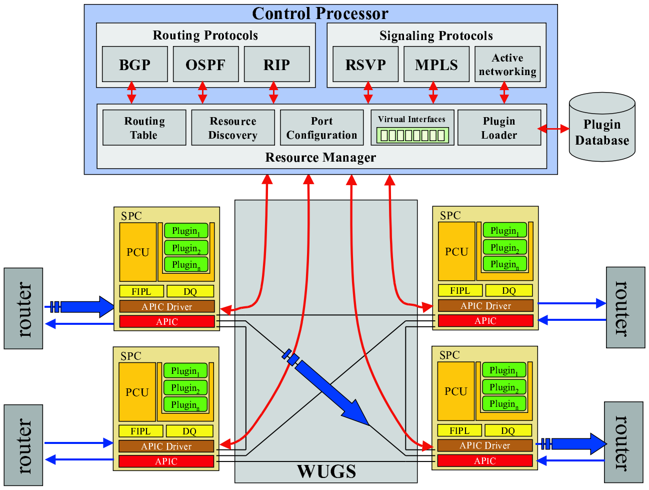

Figure 4: MSR Logical View

Figure 4 gives an alternative view of the MSR showing some of the functions along the control and data paths. This section describes the activities that involve the Control Processor (CP) and its interaction with the PPs. The software framework running on the CP supports system control, resource management, routing, and distributed queueing.

3.1. Internal Communication

Communication between the MSR's distributed components is built on top of a set of ATM virtual circuits using VCI (Virtual Circuit Identifier) allocation rules that simplify their use. Also, the CP uses the concept of *virtual interfaces* to easily segregate traffic arriving from the PPs. The VCI space partitioning separates packets into three traffic types: 1) Control, 2) IP, and 3) Native ATM.

In addition, the VCI allocation rules for inter-PP communication simplifies the identification of the sending port. A VCI is treated as a "tag" that identifies the sending port. For example, the MSR uses VCI ($i + 40$) to identify IP traffic transiting the switch from port i (i is between 0 and 7 since there are 8 ports). Any output PP knows that all packets from VCI ($i + 40$) is IP traffic from input port i. Here, 40 is the base VCI for IP traffic. Other traffic types use different base VCIs.

In certain situations, the CP will have to process a packet on behalf of a PP. For example, one implementation of `traceroute` sends UDP packets to an unused destination port[19] and determines completion when it receives a "`port unreachable`" response message from the destination. When the destination is an MSR interface, the MSR's CP processes the UDP packet. The MSR's virtual interface concept accomplishes this without the need to encapsulate the incoming packet before sending it to the CP. The PP at the MSR interface passes the packet to the CP using its unique virtual interface VCI, and the CP's OS kernel "tags" the packet upon arrival. Then, the CP sends the appropriate ICMP response on the PP's behalf (the source IP address is the IP address of the interface). In this fashion, an MSR port looks like a logical interface of the CP.

3.2. System Configuration

System initialization sets up communication paths between all processors (CP, PPs) and initializes the PPs with instructions and data. This boot process is multi-tiered. First, it performs a low-level initialization sequence so that the CP can communicate with the PPs. Next, the process discovers the number, location, and types of computing resources and links at each port.

The following sequence is executed:

Configuration: The CP controls the operation of its PPs using ATM control cells. Therefore, communication must be established between the CP and its PPs even before the discovery step can be performed. The CP sets up predefined ATM virtual circuits (VCs). These include VCs for control cells, for program loading and for forwarding IP packets from input ports to output ports.

Discovery: The CP discovers the low-level configuration of a port by sending control cells to each potential processor at each port using the VCIs in step 1. Each processor will report the characteristics of the adjacent card that is further from the switch port. The responses indicate the type of processors at each port and the link rate.

SPC Initialization: The CP downloads a NetBSD kernel and memory-resident filesystem to each SPC using a multicast VC and AAL5 frames and completes each SPC's identity by sending them their port location using an MSR control message.

FPX Initialization: Initialization of an FPX follows a similar sequence. A program and configuration is loaded into the *RAD reprogram memory* under control of the NID using control cells.

Once the last cell has been successfully loaded, the CP sends a control cell to the NID to initiate the reprogramming of the RAD using the contents of the reprogram memory.

3.3. Route Management

The MSR maintains information about other routers in the network by running Zebra [20], an open-source routing framework distributed under the GNU license. Zebra supports various interior (OSPFv3, RIPv2, RIPng) and exterior (BGP-4) routing protocols. Each individual routing protocol contributes routes to a common routing table managed by the CP. Based on this routing table, the CP computes a forwarding table for each port, which it keeps synchronized with the routing table. As routing protocols receive updates from neighboring routers that modify the routing table, the CP continuously recomputes the forwarding tables and propagates the changes to each port.

The forwarding tables stored in the FPXs and SPCs use a tree bitmap structure for fast packet classification with efficient memory usage [21]. The tree bitmap algorithm employs multibit trie data structures that are ideally suited for fast hardware implementation [7].

When the CP receives a route update from another router to add or delete a path, it creates a new internal tree bitmap structure that reflects the modified forwarding table. Then, it sends ATM control cells to the Fast IP Lookup components in the SPC or FPX representing the modifications to the multibit trie structure.

3.4. Signaling and Resource Management

The CP also handles various signaling protocols (e.g., RSVP, MPLS) and supports active networking functionality. The signalling protocols allow applications to make bandwidth reservations required for QoS guarantees. When a bandwidth reservation request arrives at the CP, the CP first performs admission control by checking for sufficient resources. If admission control succeeds, the CP reserves the required bandwidth on both the input and output ports and returns a signaling message to grant the reservation.

The MSR signaling system also supports protocols that establish flow-specific routes. Flow-specific routes might be used by applications requiring specific QoS guarantees or flows that need to transit specific network nodes in a given order for active processing.

Flow-specific routes are handled by performing the routing lookup using a combination of the destination IP address, source IP address, destination port, source port, and protocol fields. In this way, a single longest-matching-prefix lookup will handle both the normal IP routes and flow-specific routes.

In addition, the MSR provides flow-specific processing of data streams. An active flow is explicitly set up using signaling mechanisms which specify the set of functions which will be required for processing the data stream. In the context of the MSR, plugins are code modules that provide a specific processing function and can be dynamically loaded and configured at each port.

If an active signaling request references a plugin that has not been deployed on the router, the CP retrieves the plugin code from a remote code server, checks its digital signature, and then downloads it to a PP where it is configured using ATM control cells. Once the plugin has been successfully loaded and configured, the CP installs a filter in the port's forwarding table so that matching packets will be routed to the plugin.

3.5. Congestion Avoidance and Distributed Queueing

Under sustained overload, the internal links of the WUGS can become congested leading to substantially reduced throughput. Our Distributed Queueing (DQ) algorithm allows the MSR to perform like an output queueing system (switch fabric and output queues operate at the aggregate input rate) but with a switch fabric and output queues that run near the rate of a single input link [22].

Mechanism: The DQ algorithm employs a *coarse scheduling* approach in which inputs periodically broadcast information about their own backlog to each output, and outputs periodically broadcast information about their queue length and output rate back. The MSR uses *Virtual Output Queueing* [23, 24, 25] to avoid head-of-the-line blocking. Each input maintains separate queues for each output allowing inputs to regulate the flow of traffic to each of the outputs so as to keep data moving to the output queues in a timely fashion, while avoiding internal link overload.

In our implementation, each broadcast contains both input and output information since a single PP handles both input and output traffic. Each input port i uses the information to compute new input-to-output flow rates r_{ij} for each output port j. Our current testbed has eight ports and uses only SPCs as PPs. The SPC uses a different VCI to transmit packets to each output port and can set the output rate on a per VCI basis using the APIC's pacing facility. After each flow rate update period, each SPC updates the APIC pacing rate accordingly.

Algorithm: At every update period (currently 100 μsec), each input port i recalculates the rate $r_{i,j}$ at which it can send traffic to output j, using:

$$r_{i,j} = min(\frac{L_j B_{i,j}}{B_j + \sum_h B_{h,j}}, \frac{S B_{i,j}}{\sum_h B_{i,h}})$$

where L_j is the link bandwidth at output j, B_j is the backlog at output j, $B_{i,j}$ is the total traffic queued at input i destined to output j and S is the internal speed of the switching fabric. The first term shares the output bandwidth proportionally between the inputs and output j. By including the output backlog (B_j) in the denominator, traffic will be directed at outputs with smaller backlogs. The second term shares the switch fabric bandwidth proportionally between the inputs. The minimum over the two expressions insures that inputs can always send at their assigned rates without overloading the switch fabric; i.e., $(\sum_h r_{i,h} \leq S)$.

4. Port-Level Processing

4.1. IP Processing

This section describes IP packet processing and the programmable network environment in the SPC. When the SPC is the only PP at a port, it must handle all input and output processing. Although ideally every port should have both an FPX to handle the typical case (e.g., no active processing or options) and an SPC to handle special cases (e.g., active processing), it is desirable to have an SPC that has full port functionallity for several reasons:

- *Rapid Prototyping*: A prototype MSR testbed can be constructed even though the FPX is still under development.
- *Lower Cost*: A lower cost (but slower) MSR can be constructed using only SPC PPs.
- *Measurement and Experience Base*: Experience with the SPC may be fruitful in the development of the FPX, and experimental features can be examined using the SPC as a preliminary step to commiting to hardware. Furthermore, the acceleration benefits of using the FPX can be quantified.

In order to reduce overhead, the IP data path and basic resource management functions have been completely incorporated into the APIC interrupt handler. Exceptions include the allocation of MSR specific memory objects for buffers and the scheduling of periodic tasks.

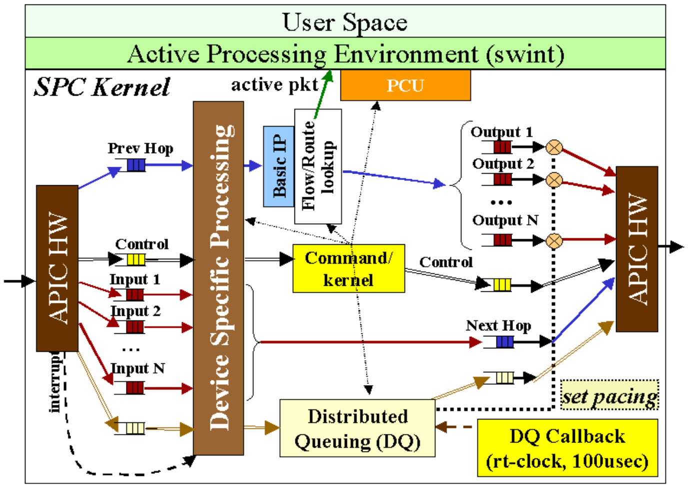

Figure 5: IP Processing on the SPC

The main areas of improvement are:

- IP data path selection (input versus output processing);
- Receive and send packet buffer management;
- APIC descriptor chain processing; and
- Interaction with the APIC hardware (reading/writing across the PCI bus).

Figure 5 shows the data paths through the SPC kernel as it forwards IP packets (input and output side), processes distributed queuing updates or responds to control cells from the CP.

Code Path Selection: As indicated in Section 3.1, VCIs are used as demultiplexing keys for incoming packets. The VCI of an incoming packet indicates to the kernel whether it is from a previous hop router, the CP or one of the connected MSR input ports. If received from a previous hop, the packet is sent to the input port processing code: basic IP processing, IP lookup and then one of the MSR's virtual output queues. If received from an input port, the packet can be immediately sent to the next hop router or endsystem [1].

APIC Processing and Buffer Management: The operation of an APIC reduces the load on the SPC by asynchronously performing sequences of read/write operations described by *descriptor chains*. An APIC descriptor is a 16-byte structure that describes the data buffer to be written to for receive, or read from transmit. During initialization a contiguous chunk of memory is allocated for the descriptors (half for TX (transmit) and half for RX (receive)). The driver and APIC hardware then use a base address and index to access a particular descriptor.

During initialization, another contiguous region of memory is allocated for IP packet buffers. Each buffer is 2 KB, and there are an identical number of buffers and RX descriptors. Each buffer is bound to an RX descriptor such that their indexes are the same. Consequently, given a descriptor address or index, the corresponding RX buffer can be located simply and quickly. The reverse operation from buffer to descriptor is equally fast. This technique makes buffer management trivial, leaving only the management of the RX descriptor pool as a non-trivial task.

Since there are the same number of TX descriptors as RX descriptors, we are always guaranteed to be able to send a packet once it is received. Note that when sending a packet, the receive buffer is bound to the TX descriptor. The corresponding RX descriptor is not available for reuse until the send operation completes. This has the nice effect that the SPC will stop receiving during extreme overload and avoid unnecessary PCI and memory traffic and receiver livelock.

4.2. Programmable Networks Environment

On each port of the MSR, the SPC runs a modified NetBSD kernel that provides a programmable (plugin) environment for packet processing (Figure 6). The SPC environment includes functionality to support both traditional IP forwarding as well as flow-specific processing.

Traditional IP forwarding includes controlling the APIC hardware, packet classification and fair output queuing which runs at the hardware interrupt level. Active packet processing is handled at the lower priority software interrupt level. Therefore, active processing can be preempted by APIC device interrupts associated with packet arrivals, guaranteeing that packets can be immediately sent and received from the hardware.

If an MSR port is equipped with an FPX, packets are classified using the hardware implementation of the Fast IP Lookup (FIPL) algorithm and sent to the SPC on a special VCI, signaling the SPC

[1] Currently the MSR only supports one directly connected device. However we plan to extend this to some finite number of connected hosts and routers.

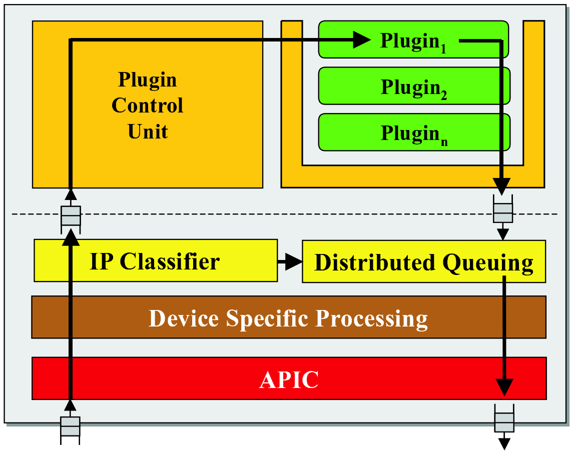

Figure 6: SPC Plugin Environment

that the packet is already classified. If no FPX is present at a port, the packet arrives at the standard VCI and the SPC performs the lookup itself using a software implementation of the IP classifier.

Regardless of how the packet is identified as requiring active processing, it is enqueued into the active processing queue and control is given to the Plugin Control Unit (PCU).

The PCU provides an environment for loading, configuring, instantiating and executing plugins. Plugins are dynamically loadable NetBSD kernel modules which reside in the kernel's address space. Since no context switching is required, the execution of plugins is highly efficient.

For the design of plugins, we follow an object-oriented approach. A *plugin class* specifies the general behavior of a plugin and defines how it is initialized, configured and how packets need to be processed. A *plugin instance* is a runtime configuration of a plugin class bound to a specific flow. It is desirable to have multiple configurations of a plugin, each processing its specific flow and having its own data segment that includes the internal state. Multiple plugin instances can be bound to one flow, and multiple flows can be bound to a single instance.

Through a virtual function table, each plugin class responds to a standardized set of methods to initialize, configure and process plugins. All code for initialization, configuration and processing is encapsulated in the plugin itself. Therefore, the PCU is not required to know anything about a plugin's internal details.

When the PCU is in control, it decides which packet needs to be scheduled next for processing in order to meet each flow's QoS delay guarantees. Suitable execution scheduling algorithms have been discussed in several references [26, 27].

Once a packet has been dequeued from the active processing queue, the plugin environment invokes the processing function of the corresponding plugin instance, passing it a reference to the packet to be processed. The processing might alter the packet payload as well as the header. If a packet's destination address has been modified, the packet needs to be reclassified since the output port might have changed before the packet is finally forwarded.

5. Measurement Experiments

In this section we focus on the router throughput when using SPCs as both input and output port processors. In particular, we measure the packet forwarding rate and data throughput for different IP packet sizes.

5.1. Experimental Setup

Figure 7 shows the experimental setup used for our tests. The configuration includes an MSR with CP and one PC on port P4 acting as a traffic source. The ATM switch core is an eight-port WUGS configured with an SPC on each port. The CP and traffic source are both 600 MHz Pentium PCs with APIC NICs.

The experiments use four key features of the WUGS: input port cell counters, a calibrated internal switch clock, and ATM multicast and cell recycling. The CP reads the cell counter from the

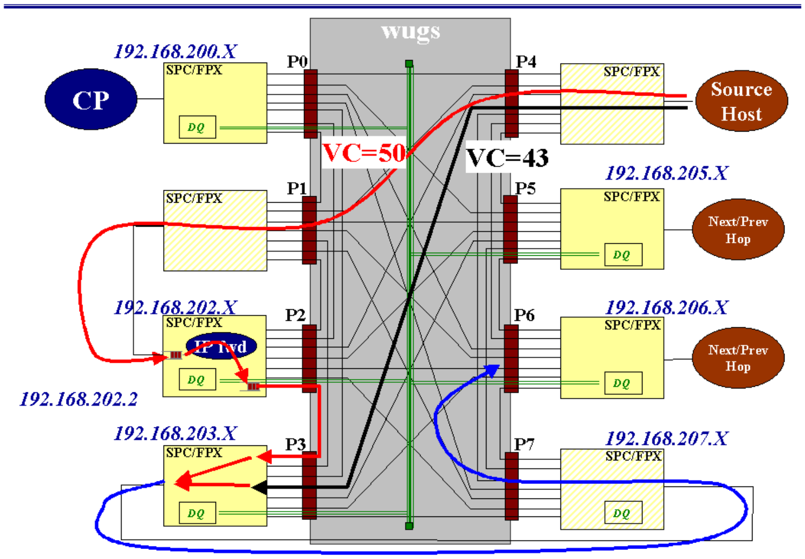

Figure 7: Experimental Setup

switch input ports and uses the switch cell clock to calculate a cell rate. The packet rate can easily be derived from the cell rate since the number of cells per packet is a constant for an individual experiment.

The multicast and recycling features of the WUGS were used to amplify the traffic volume for single-cell IP packets. Cell traffic can be amplified by 2^n by copying and recycling cells through n VCIs before directing the cells to a target port. However, this feature can not be used for multi-cell IP packets since the ATM switch core does not prevent the interleaving of cells from two packets.

The SPCs on ports P2 and P3 were configured to operate as IP packet forwarders. Port P2 is used as the input port and port P3 as the output port. All other SPCs are disabled so that traffic will pass through them unaffected.

Typically, hosts or other routers would be connected to each port of an MSR. However, to facilitate data collection we have directly connected the output of port P1 to the input of port P2 and the output of port P3 to the input of P7. Our data source is connected to port P4. Thus we can use:

- The cell counters at port P4 to measure the sending rate;
- The cell counters at port P2 to measure the traffic forwarded by the input side PP at port P2; and
- The cell counters at port P7 to measure the traffic forwarded by the output side PP at port P3.

IP traffic is generated by using a program that sends specific packet sizes at a prescribed rate. Packet sending rates are controlled using two mechanisms: 1) logic within the traffic generator program, and 2) for high rates, the APIC's pacing facility. These two mechanisms produced both high and consistent sending rates.

5.2. Small-Packet Forwarding Rate

In order to determine the per packet processing overhead, we measured the forwarding rate of 40-byte IP packets (1 ATM cell each) at the input and output ports. Single-cell packet rates as high as 907 KPps (KiloPackets per second) were generated by using the ATM multicast and cell recycling features of the switch to multiply the incoming traffic by a factor of 8.

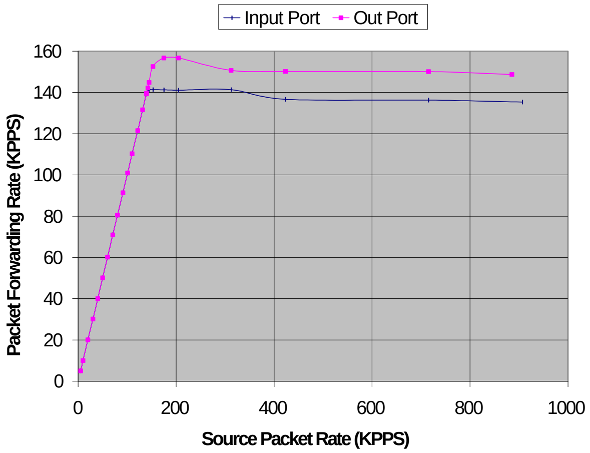

Figure 8: Packet Forwarding Rate for 40-Byte IP Packets

Figure 8 shows the packet forwarding rate for 40-byte IP packets. The line labeled “Input Port” represents the packet forwarding rate for an SPC operating as an input PP. Similarly, the line labeled “Output Port” is the corresponding rate on the output side.

The maximum forwarding rate at an input port PP is about 140 KPps. As expected, the output port PP has a higher forwarding rate since it does not perform IP destination address lookup. Furthermore, 140 KPps is sustained even for a source rate as high as 900 KPps. This rate stability at high loads is a consequence of our receiver livelock avoidance scheme. The throughput can be obtained from Figure 8 by multiplying the packet forwarding rate by the packet size (40 bytes). A calculation would show that 140 KPps corresponds to a maximum forwarding data rate of around 45 Mbps and to a packet processing time of approximately 7.1 μsec.

5.3. Throughput Effects of Packet Size

We next measured the effect of the packet size on the packet forwarding rate. Because IP packets larger than 40 bytes require more than one cell (there is 8 bytes of overhead), we no longer used ATM multicast with cell recycling to amplify the traffic. We used a single host to generate traffic using packet sizes ranging from 40 bytes to 1912 bytes.

Figure 9 shows the packet forwarding rate as a function of the input packet rate for a range of packet sizes. A straight line with a slope of 1 corresponds to the case when there are no bottlenecks along the path through the router. For all packet sizes, the forwarding rate starts out as a line with slope 1 until finally a knee occurs ending in a horizontal line. The horizontal portion of a forwarding

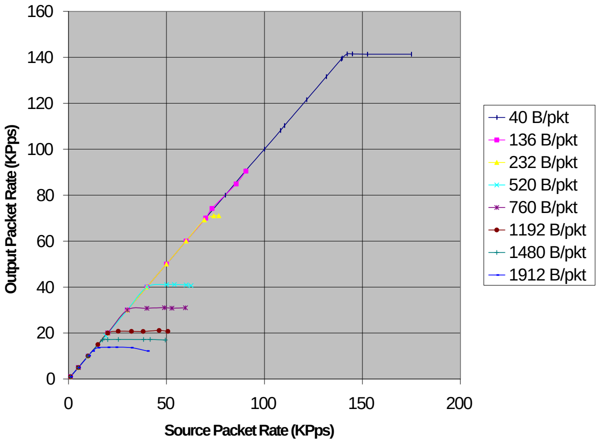

Figure 9: Packet Forwarding Rate for Various IP Packet Sizes

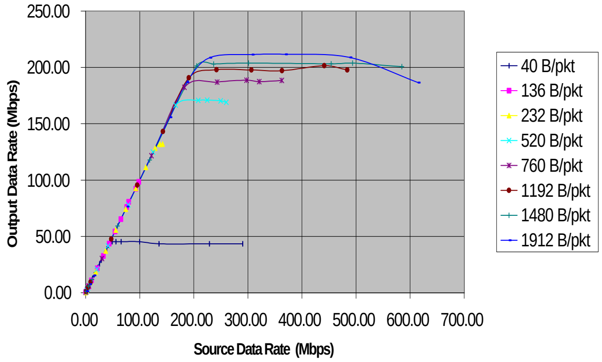

Figure 10: Router Throughput for Various IP Packet Sizes

rate curve is an indication of CPU, PCI bus and memory system bottlenecks. Saturation occurs earlier (smaller packet source rate) for larger packets since more memory and PCI bus bandwidth is being consumed.

Figure 10 shows the output data rate which can be derived from Figure 9 by multiplying the packet forwarding rate by the packet size.

5.4. Analysis of Results

In analyzing our performance results, we considered three potential bottlenecks:

- PCI Bus (33 MHz, 32 bit)
- SPC Memory Bus (66 MHz EDO DRAM)
- Processor (166 MHz Pentium MMX)

Each of these comes into play at different points in the packet forwarding operation.

Our studies of the PCI bus operations that take place to forward a packet indicate that there are 3 PCI read operations and 5 PCI write operations which together consume 60 bus cycles. Additionally, under heavy load, we can expect 64 wait cycles to be introduced. Thus a total of 124 bus cycles (30.3 ns per cycle on a 33 MHz bus) or 3.72 μsec are consumed by PCI bus operations in the forwarding of a packet.

The average software processing time for performing the simple IP lookup scheme utilized in our test cases has been measured to be 3.76 μsec. This, combined with the PCI bus time calculated above gives us a per packet forwarding time of 7.48 μsec. This is very close to the time of 7.1 μsec for the small packet forwarding rate of 140 KPps shown in Figure 8.

One of the problems that we have to overcome with the SPC is a bug in the APIC chip. This bug causes the received word order on an Intel platform to be incorrect. In order to work around this, the APIC driver must perform a word swap on all received data. Thus, each received packet may cross the memory bus 4 times:

- APIC writes packet to memory
- CPU reads packet during word swapping
- CPU writes packet during word swapping
- APIC reads packet from memory

We have demonstrated the impact of the word swapping by eliminating most of it from a simple test using 1912-byte packets. In this test, we only performed the word swap on the 20 bytes of the IP header so that we could perform our IP lookup operation. In this test case, our forwarding rate increased from about 14 KPps to 22 KPps, a 50% increase. The corresponding throughput increased from 212 Mbps to 336 Mbps.

6. Concluding Remarks

Additional performance measurements of the MSR are in progress, and a number of developments and extensions are underway. First, the *integration of the FPX* with the current MSR configuration will soon commence. The Fast IP Lookup (FIPL) algorithm has been implemented in reprogrammable hardware using the FPX, and simulations have demonstrated a speed of over nine million lookups per second may be possible on each port. In addition, other applications are currently being ported to the FPX. Second, *SPC II* is under development with availability planned for the end of 2001. It will have a faster processor (500 MHz to 1 GHz PIII), much higher main memory bandwidth (SDRAM), and a larger memory (256 MB). Third, a *Gigabit Ethernet line card* is being designed around the PMC-Sierra PM3386 S/UNI-2xGE Dual Gigabit Ethernet Controller chipset with plans for availability in early 2002. This will allow us to interface the MSR to routers and hosts that have Gigabit Ethernet interfaces. Fourth, many *CP software components* are in their early prototyping stage. Some of these components include: 1) Automatic multi-level boot process that starts with discovery and ends with a completely configured, running router; 2) Network monitoring components based on active, extensible switch and PP MIBs and probes provides a multi-level view of the MSR router; and 3) the Zebra-based routing framework.

The Washington University MSR provides an open, flexible, high-performance router testbed for advanced networking research. Its parallel architecture will allow researchers to deal with many of the same real design issues faced by modern commercial designers. Finally, its reprogrammability in combination with its open design and implementation will make it an ideal prototyping environment for exploring advanced networking features.

References

[1] T. Chaney and A. Fingerhut and M. Flucke and J. S. Turner, “Design of a Gigabit ATM Switch,” in *IEEE INFOCOM ’97*, (Kobe, Japan), IEEE Computer Society Press, April 1997.

[2] J. S. Turner and A. Staff, “A gigabit local atm testbed for multimedia applications,” Tech. Rep. ARL-94-11, Applied Research Laboratory, Washington University in St. Louis, 1994.

[3] A. T1.106-1988, *Telecommunications - Digital Hierarchy Optical Interface Specifications: Single Mode*, 1988.

[4] H.-P. Corporation, “Hdmp-1022 transmitter/hdmp-1024 receiver data sheet,” 1997.

[5] S. Choi, J. Dehart, R. Keller, J. W. Lockwood, J. Turner, and T. Wolf, “Design of a flexible open platform for high performance active networks,” in *Allerton Conference*, (Champaign, IL), 1999.

[6] D. S. Alexander, M. W. Hicks, P. Kakkar, A. D. Keromytis, M. Shaw, J. T. Moore, C. A. Gunter, J. Trevor, S. M. Nettles, and J. M. Smith in *The 1998 ACM SIGPLAN Workshop on ML / International Conference on Functional Programming (ICFP)*, 1998.

[7] 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.

[8] 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.

[9] D. E. Taylor, J. W. Lockwood, and N. Naufel, “Generalized RAD Module Interface Specification of the Field-programmable Port eXtender (FPX),” tech. rep., WUCS-01-15, Washington University, Department of Computer Science, July 2001.

[10] J. W. Lockwood, “Evolvable internet hardware platforms,” in *The Third NASA/DoD Workshop on Evolvable Hardware (EH’2001)*, pp. 271–279, July 2001.

[11] Z. D. Dittia, “ATM Port Interconnect Chip.” www.arl.wustl.edu/apic.html.

[12] “NetBSD.” http://www.netbsd.org.

[13] Intel Corporation, *Intel 430HX PCISET 82439HX System Controller (TXC) Data Sheet*. Mt. Prospect, IL, 1997.

[14] Intel Corporation, *822371FP (PIIX) and 82371SB (PIIX3) PCI ISA IDE Xcelerator Data Sheet*. San Jose, CA, 1997.

[15] Xilinx, Inc., *Xilinx 1999 Data Book*. San Jose, CA, 1999.

[16] J. D. DeHart, W. D. Richard, E. W. Spitznagel, and D. E. Taylor, “The Smart Port Card: An Embedded Unix Processor Architecture for Network Management and Active Networking,” Department of Computer Science, Technical Report WUCS-01-18, Washington University, St. Louis, 2001.

[17] Z. D. Dittia, J. R. Cox, Jr., and G. M. Parulkar, “Design of the APIC: A High Performance ATM Host-Network Interface Chip,” in *IEEE INFOCOM ’95*, (Boston, USA), pp. 179–187, IEEE Computer Society Press, April 1995.

[18] Z. D. Dittia, G. M. Parulkar, and J. R. Cox, Jr., “The APIC Approach to High Performance Network Interface Design: Protected DMA and Other Techniques,” in *Proceedings of INFOCOM ’97*, (Kobe, Japan), pp. 179–187, IEEE, April 1997.

[19] W. R. Stevens, *TCP/IP Illustrated, Volume 1*. Reading, Massachusetts: Addison Wesley, 1993.

[20] The Zebra Organization, “GNU Zebra.” http://www.zebra.org.

[21] W. Eatherton, “Hardware-Based Internet Protocol Prefix Lookups,” Master’s thesis, Department of Electrical Engineering, Washington University in St. Louis, 1998.

[22] E. Leonardi, M. Mellia, F. Neri, and M. A. Marsan, "On the stability of input-queued switches with speed-up," *IEEE Trans. on Networking*, vol. 9, pp. 104–118, Feb 2001.

[23] T. Anderson, S. Owicki, J. Saxe, and C. Thacker, "High speed switch scheduling for local area networks," *ACM Trans. on Computer Systems*, vol. 11, pp. 319–352, Nov 1993.

[24] N. McKeown, V. Anantharam, and J. Walrand, "Achieving 100input-queued switch," *IEEE Trans. Communication*, vol. 47, pp. 1260–1267, 1999.

[25] N. McKeown, M. Izzard, A. Mekkittikul, W. Ellersick, and M. Horowitz, "The tiny tera: A packet switch core," in *Hot Interconnects*, 1996.

[26] T. Wolf and D. Decasper, "CPU Scheduling for Active Processing using Feedback Deficit Round Robin," in *Allerton Conference on Communication, Control, and Computing*, (Monticello, Illinois), September 1999.

[27] P. Pappu and T. Wolf, "Scheduling processing resources in porgrammable routers," Submitted to IEEE Infocom 2002.