Design of a Flexible Open Platform for High Performance Active Networks

Sumi Choi, Dan Decasper, John Dehart, Ralph Keller
John Lockwood, Jonathan Turner and Tilman Wolf
{syc1, dan, jdd, keller, lockwood, jst, wolf}@arl.wustl.edu

Abstract

This paper describes an architecture for a high performance active router. The system is designed as an open research platform, with a range of configuration options and possibilities for extension in both the software and hardware dimensions. The system is built around a scalable switch fabric and includes a general-purpose processor subsystem at each port, enabling flexible packet processing and deployment of flow-specific *active plugins*. Such a research platform is becoming an indispensable tool for effective systems research in networking and distributed systems.

1 Introduction

In the eighties and early nineties, networking research has concentrated largely on how to improve network performance, by designing switches and routers that could handle large numbers of gigabit rate links. This research has led to the development of systems capable of supporting 2.4 Gb/s links and system capacities in the 100 Gb/s range. Systems now being developed feature scalable multistage interconnection networks and custom hardware processing enabling them to support gigabit links and capacities exceeding 1 Tb/s. While there is a continuing need to improve performance, it appears likely that future developments will be driven primarily by continuing improvements in the underlying electronics technology, accompanied by incremental refinements in system architectures.

Advances in electronics technology affect switches and routers in two ways. First, as the IO capacity of individual integrated circuits grows, it becomes possible to process data at higher rates in a single chip, allowing support of higher speed links and/or more economical processing of data from lower speed links. Second, as the logic capacity of integrated circuits grows, it allows more complex processing to be performed within a single chip. Because the logic capacity of integrated circuits is growing considerably faster than their IO bandwidth, there is a real opportunity to perform more sophisticated processing, without compromising overall system performance. These fundamental trends are already evident in the growing sophistication of switches and routers, which now commonly employ complex queueing and packet filtering mechanisms.

A natural next step in this evolution is the incorporation of specialized programmable components within switches and routers to allow more flexible processing than is possible with custom hardware. Emerging products [3] highlight this direction, providing tens of on-chip processors, enabling gigabit packet processing rates. While the memory limitations of the current generation of devices significantly constrains the range of applications to which

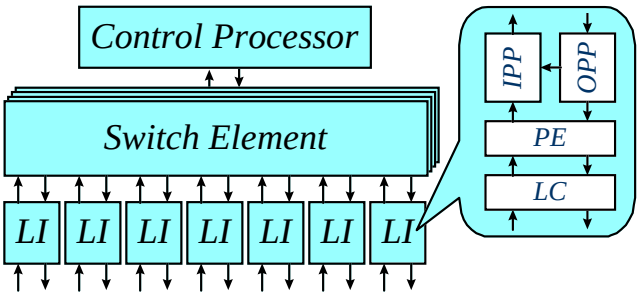

Figure 1: System Organization of Active Router

they can be applied, they clearly show the direction in which the technology is now moving. The introduction of programmable processors with substantial amounts of memory will enable richer processing possibilities, including the embedding of application processing within networked devices. Active networking [13] offers one of the more intriguing visions for how such embedded processing capabilities might be invoked on behalf of users, but it is by no means the only option.

This paper describes an open and extensible research platform for studies in active networking and related areas of networking research. We describe the key hardware and software components making up the system, with an emphasis on the different ways these components can be used. The platform allows new components to be easily introduced, either replacing or augmenting existing components. Such a research platform provides a valuable tool for systems researchers in networking and distributed computing. Because the internal technical specifications are public, the system can be used in innovative ways or modified to enable the provision of new capabilities. Most of the components making up this platform have been distributed to about 30 different universities as part of an NSF-sponsored initiative, and all of the components are now, or soon will be, generally available to the research community [12, 14].

2 System Organization

Figure 1 shows the overall organization of the *Washington University Gigabit Router* (WUGR). At the core of the system is an ATM switch built around a scalable fabric constructed from eight port *Switch Elements* (SE). The base configuration shown in Figure 1 contains a single SE (implemented using four integrated circuits), but the same chips can be used to construct multistage interconnection networks with tens or hundreds of gigabit links [1]. The switch fabric connects to the external links through a set of *Link Interfaces*. Each LI contains an *Input Port Processor* (IPP), an *Output Port Processor* (OPP), a *Processing Element* (PE) and a *Line Card* (LC). The IPPs perform the ATM routing lookup on received cells and the OPPs buffer cells awaiting transmission on the external link. The system supports both virtual path and virtual circuit switching, and both one-to-many and many-to-many multicast. The chip set was originally intended to support link speeds of up to 2.4 Gb/s; however, the current implementation is limited to link speeds of 1 Gb/s. Future versions of the chip set are expected to correct this.

Connections to external links are provided through *Line Cards* (LC) that implement the opto-electronic conversion needed for transmission over fiber optic links and transmission encoding and decoding. Several different line cards have been implemented, including line cards for SONET links at OC-3 and OC-12 rates. The most common line card provides 1 Gb/s transmission rates using an optical data link format similar to Fibre Channel.

Each port on the router has a *Processing Element* card which connects between the

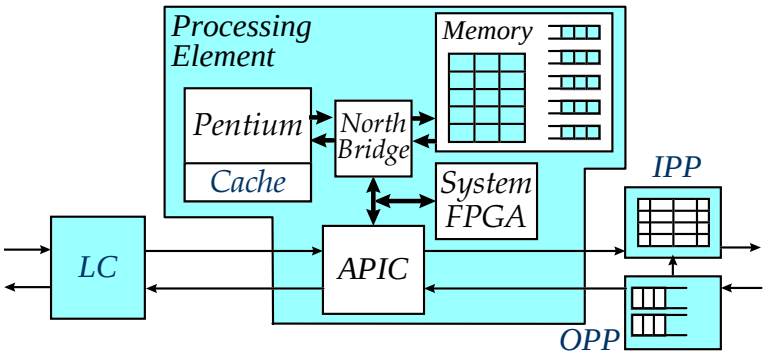

Figure 2: Details of the Processing Element

IPP/OPP and the Line Card, as shown in Figure 2. The PE contains an embedded processor module comprising a Pentium microprocessor (167 MHz), secondary cache, and North Bridge chip, which controls access to memory and which interfaces to the PCI bus used to access peripheral devices. The PE card includes 64 MB of DRAM, a dual port network interface chip for accessing data passing in and out of the system and a *System FPGA*, which implements an interrupt controller, some basic timer mechanisms and miscellaneous other functions that are usually provided by a collection of discrete components in an ordinary PC. The PE card also provides two serial interface ports that can be connected to a computer to assist in debugging the embedded software.

The dual port network interface chip is called the APIC (ATM Port Interconnect Controller). It has two 16 bit Utopia interfaces and a PCI bus interface that can be operated in either 32 bit or 64 bit mode (the PE board uses it in 32 bit mode). The APIC has an on-chip virtual path & circuit translation table and has 256 queues for virtual paths or circuits going to the PCI bus. It pipelines data directly between the network and the processor's memory, avoiding an on-chip packet buffering step, and it supports zero copy transport protocol implementations for maximum performance. The APIC also supports paced transmission of cells from each of up to 256 channels. More details can be found in [8].

The system is managed by a Control Processor (CP), which is implemented using a conventional PC with ample memory and disk. The CP implements routing protocols and other system-wide functions and communicates with the PEs through the switch. It also provides signalling and management interfaces, allowing the system to be embedded in larger testbed networks.

3 Processing Element Software

The Processing Elements (PE) provide both the core packet processing functions and the processing required for active networking. To provide maximal flexibility, the PE software is built around the Net BSD Unix system [9], an open-source Unix system that runs on a variety of platforms, has excellent networking support and a well-structured kernel that is relatively easy to modify and extend.

The kernel software has been augmented using *Router Plugins* [7], a flexible and general tool for extending the functionality of the kernel's packet forwarding path. Router plugins are kernel modules that are statically configured via network management mechanisms and which are applied to particular packets, specified through a kernel-resident packet filter database, which can be dynamically modified. Router plugins have been used to implement *Active Plugins*, which are dynamically configured kernel modules, invoked

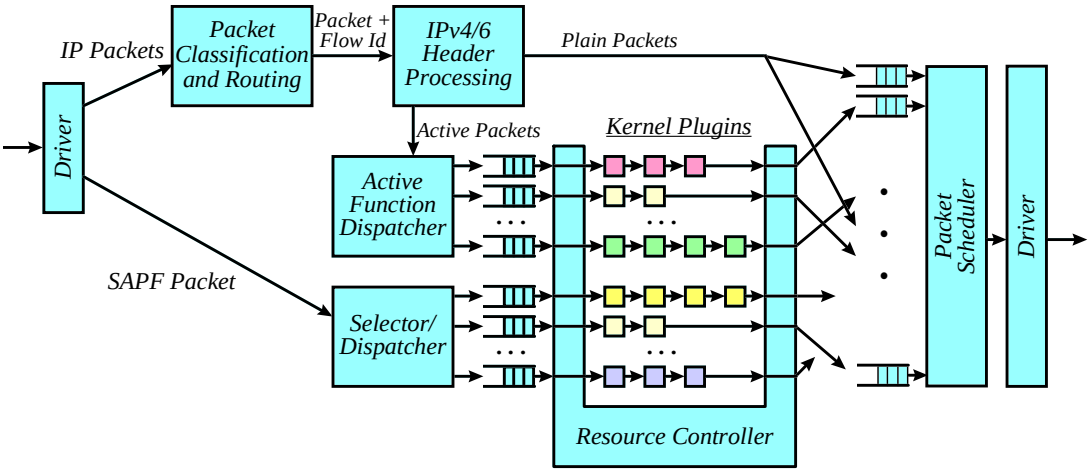

Figure 3: Processing Element Kernel Software

through plugin references carried in active packets moving through the network. The PE software, installs active plugins on-demand, as they are referenced by arriving packets. Plugins that are not available locally are retrieved from one of a set of trusted code-servers. Cryptographic techniques are used to verify that retrieved code does indeed come from a trusted server and to authenticate the identity of the software developer. A policy database is used to control the code servers that are used and/or the developers from which code is accepted. One can also restrict the application of specific plugins to specific packets or classes of packets.

Figure 3 shows the principal components of the PE kernel software involved in processing packets. Packets are assembled in buffers by the APIC and the device driver (the driver prepares the buffer data structures and the APIC transfers and checks the integrity of the data). Packets are then typically passed to a packet classification module that does a lookup based on source and destination address, source and destination port number and protocol. This lookup yields a flow identifier that is used to facilitate flow-specific processing. Packets then pass through an IP processing module that performs standard header processing functions. Non-active IP packets are then placed in one of a set of output queues.

Active packets (identified by a specific protocol value in the IP protocol field) are forwarded to an *Active Function Dispatcher* (AFD), which examines the packet header further to extract the active plugin identifiers that specify what processing that the packet requires. If the required plugins are already present in the kernel, the AFD places the packet in the appropriate queue. The *Resource Controller* (RC) passes packets to the appropriate plugins in accordance with the PE's resource allocation policy, and tracks the amount of processing time used by each packet, in order to regulate future packet selections.

If a requested plugin is not present when a packet arrives, a request is passed to the router's plugin management software. Often, the required plugin will be found in the CP's plugin cache. If not, a request is issued to one or more remote servers to supply the plugin. Upon retrieval, the plugin is configured into the kernel and can be used to process subsequent packets. A packet triggering a plugin retrieval can optionally be forwarded without active processing, queued pending plugin arrival or discarded.

The PE software supports a streamlined processing path for active packets. Packets using the *Simple Active Packet Format* (SAPF) contain a *selector* field which is filled in by the upstream router and used to to enable rapid access to flow-specific state, without the need for the relatively time-consuming packet classification step. The SAPF format also eliminates the need to parse the standard active packet header, which involves extensive

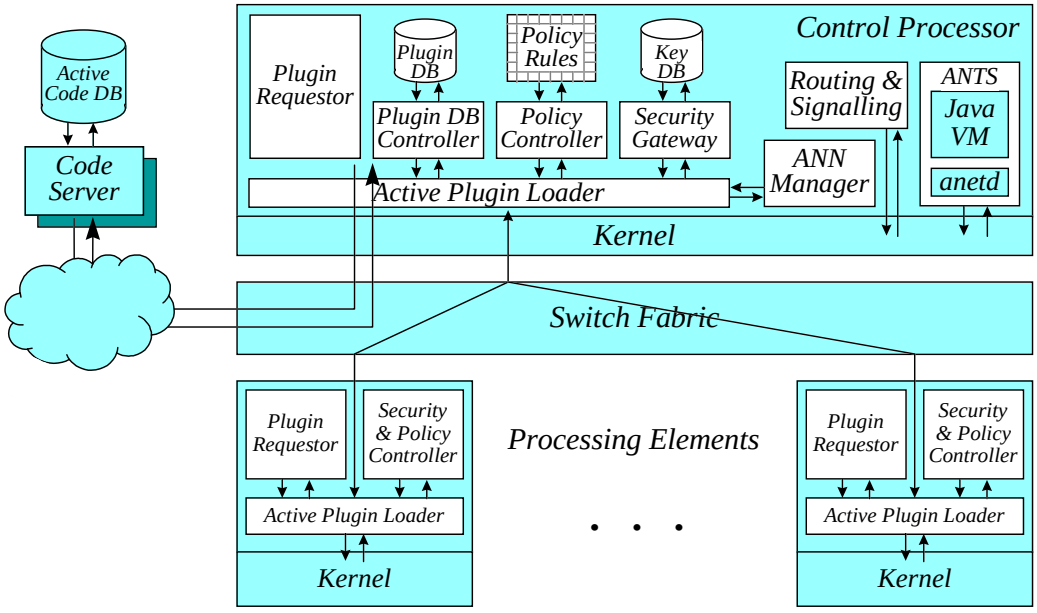

Figure 4: System Level Software Architecture

option processing. The use of the SAPF format is triggered dynamically, using a process similar to *tag switching* [2].

4 System Level Software Architecture

Figure 4 shows the main elements of the overall system software architecture, emphasizing the components involved in plugin management. At the bottom of the figure are PEs, which reside at the switch ports. The Control Processor software is shown at the top of the figure.

When a packet processed by a PE references a plugin that is not already present in the PE's kernel, a request is made to the PE's *Plugin Requestor*, and then forwarded through the switch to the CP, where it is delivered to the CP's Plugin Requestor. The CP's Plugin Requestor checks to see if the plugin is already available in the local Plugin Database, which caches plugins that have been recently used. If the plugin is available locally, it is returned to the requesting PE which loads the plugin into the kernel and continues processing. If the requested plugin is not available locally, a request is sent to a remote code server which responds with the requested plugin, if it is available. Code servers can be arranged into a multi-level hierarchy, and requests that can't be satisfied by one code server are passed up the hierarchy. This allows plugins to migrate from the top of the hierarchy down to the routers that need them.

When the CP receives a plugin, it authenticates the identity of the code server that sent the plugin and the software developer that created the plugin. The policy database allows the router's administrator to specify what code servers and software developers can be "trusted." It also allows limitations to be placed on how plugins can be used (for example, the use of a plugin might be limited to traffic between two specified subnetworks).

As shown in Figure 4, the CP software also includes an *Active Network Node Manager* which enables a router administrator to install a plugin from a remote location. The CP software also includes software to execute network routing and signalling protocols (e.g. RSVP) and software to implement the *Active Node Transfer System* (ANTS) environment, a popular active networking software system that can be used for management and control applications [15].

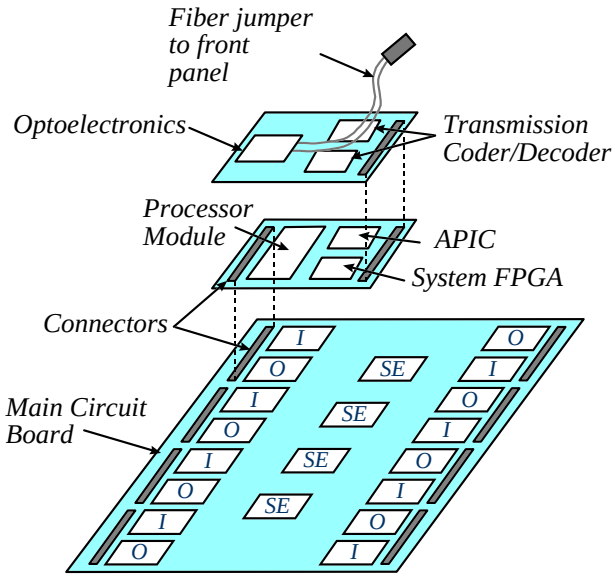

Figure 5: Physical Arrangement of Principal Physical Components

5 Configuration Options

Figure 5 shows the physical arrangement of the different components of the base configuration of the Washington University Gigabit Router. All of these components are packaged within a rack-mountable unit, approximately 12 cm tall, along with a power supply and fans. Links are connected through connectors on the front panel of the unit, which also includes indicator LEDs showing the status of each link.

The core ATM components are contained on a main circuit board that is mounted flat in the bottom of the unit. This board provides eight individual connectors that are used to connect to the PE boards, which are located directly above the main circuit board. These in turn have connectors for the line cards, which are then connected to the front panel of the unit, using fiber optic jumpers. PE cards can be stacked two to a port; or a port can be configured without a PE card, with the Line Card connecting directly to the main board. This is commonly done for the Control Processor port in the base configuration.

The base configuration provides a great deal of flexibility, allowing new components to be introduced at switch ports by creating new cards with the desired functionality and stacking them, as is done with the PEs and the existing line cards. Such cards can use either a 16 bit or 32 bit Utopia interface, although some (including the PE and some line cards) support only the 16 bit interface. While the current line cards support ATM links, it is straightforward to implement line cards for IP-over-SONET or Gigabit Ethernet interfaces as well.

The virtual circuits provided by the ATM core support a variety of configuration possibilities. In the simplest router configuration, there is a single permanent virtual circuit from each input port to each output port. In such a configuration, the PE at an input port forwards the cells of a packet on the appropriate VC. The core ATM components support two priority classes, which can be exploited by a router to provide preferential treatment to some packets (allowing it to support a real-time service with reserved bandwidth, for instance). To exploit this, the system would be configured with two PVCs from each input to each output, one of each priority. Multicast virtual circuits can also be configured, allowing multicast packets to be forwarded to a subset of outputs without the software copying. The ATM layer virtual circuits also provide a convenient mechanism for configuring signaling and network management channels that tunnel through the APICs

in the PEs and get forwarded to the CP for processing.

6 Virtual Output Queueing

The bursty nature of IP data traffic makes queueing a central issue in any router. Management of queueing is even more important in an active router, since the bandwidth requirements of packet flows can change as they move through an active network (as a result of active compression or congestion control mechanisms, for example).

While the ATM core of the WUGR router provides some internal cell buffering, the substantially larger memories of the PEs makes them the natural place for queueing in the system. Since queueing is controlled by software, a variety of strategies is possible. Here we describe one general approach as an example of the kind of possibilities that exist.

To avoid congestion within the router, one needs either per output queues that have an input bandwidth substantially larger than the link bandwidth, or one needs *virtual output queues.* A router using virtual output queues, maintains separate queues at each input port for each output port. The rates at which packets are sent from these queues are regulated so as not to overload the output links. To provide approximately the same queueing behavior as a router with true per output buffers, the different input queues sending data to a given output queue should send at a rate which is proportional to the rate at which data enters their queues.

The implementation of per output queueing requires coordination of the sending rates by different inputs to a given output. In small-to-medium sized routers, this can be accomplished by having the PEs at inputs periodically broadcast the rates at which they are receiving data for each output. Using this information, the PE at an output can determine appropriate sending rates for each input and periodically broadcast allowed sending rates for each input. The PEs at an input can then modify their sending rates using the APIC's hardware cell pacing mechanism.

For this approach to work, the control overheads must be acceptably small. If PEs broadcast their requested and allowed rates, say every 100 μs, then each will receive 2 cells every $100/n$ μs where n is the number of ports. For $n \leq 10$, this leads to an acceptably small overhead, if the control messages are processed within the APIC driver. Since a sending period of 100 μs also implies up to 200 μs of delay before a change in input rate is reflected in an increased sending rate, it is important to understand the impact of such a delay. For 1 Gb/s links, we receive at most 25 Kbytes in 200 μs. Since PEs can be configured with several megabytes of buffer space, the delay in adjusting rates is tolerable. Indeed, this example suggests that PEs could use a transmit period as long as a millisecond before the delays in adjusting rates become problematical.

7 Extension for Programmable Hardware

Modern FPGA technology has recently crossed a threshold that enables the rapid prototyping of complex hardware subsystems that once required application-specific integrated circuits. Current devices provide 200 to 300 thousand logic gate equivalents plus 100 to 200 thousand bits of static RAM. It is expected that these complexity levels will double in the next six months. Clock rates of 50-100 MHz can be achieved in complex designs and components with over 500 IO pins are available. Moreover, the design process for FPGAs is substantially simpler and less expensive, than that for ASICs making them ideal for the construction of research prototypes.

These developments open up a rich set of possibilities for hardware extensions to the WUGR router. We are currently developing a *Field Programmable Port Extender* (FPX)

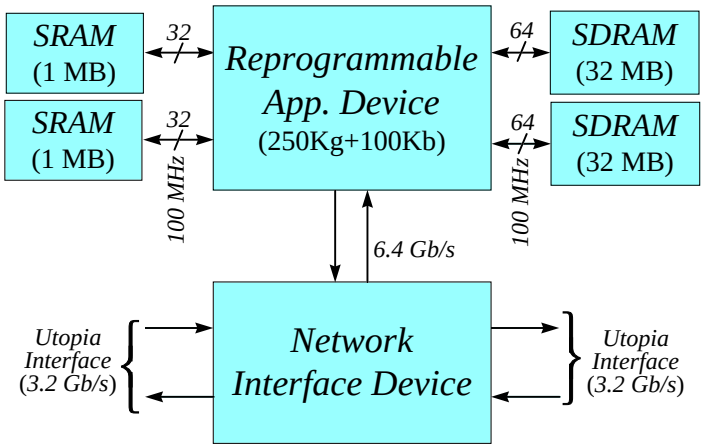

Figure 6: Field Programmable Port Extender

card which will allow new hardware functionality to be added to the system. The FPX will be a stackable card, similar to the PE card. Systems can be configured with one or more FPX cards per port and can mix PE and FPX cards on the same port (subject to constraints on physical space, power and cooling).

The planned configuration of the FPX is shown in Figure 6. The *Network Interface Device* (NID) provides access to the cell stream, much as the APIC does for the PE card. The *Reprogrammable Application Device* (RAD) provides the logic for more complex data processing. It hosts two SRAM and two SDRAM interfaces. The SRAM interfaces are both 32 bits wide and can be operated at a clock rate of 100 MHz. The SDRAM interfaces are 64 bits wide and also operate at 100 MHz. This gives and aggregate memory bandwidth of 2.4 GB/s. This arrangement makes it possible to reprogram the RAD as the system is operating and continuing to forward cells through the NID. While both parts will be reprogrammable, we expect the NID to be relatively static.

The FPX makes it possible to extend the system in a variety of ways. First, IP address lookup and packet classification can be implemented in hardware, relieving the PE software of this time-consuming task. The FPX can also separate packets requiring complex processing by the PE from packets that can be processed entirely by the FPX. This eliminates the bandwidth consumed by such packets on the PE's PCI bus, one of its more constrained resources. The FPX can also implement queueing for ordinary IP packets and can participate in the management of virtual output queues.

The most intriguing possibilities for the FPX involve the implementation of *active hardware plugins*. Such plugins could be used to implement functions that may be too computationally intensive for the PE's general-purpose hardware (encryption, video transcoding). However, implementing this vision requires the solution of a host of thorny issues, including management of the FPGA's hardware resources, coordinating the transition between successive hardware configurations and ensuring that the dynamic installation of one plugin doesn't interfere with the operation of existing plugins.

8 Closing Remarks

The continuing rapid advances in cost/performance of electronics technology is making it possible for network elements to provide ever-more sophisticated processing of data, as it moves through the internet. One of the key challenges for networking and distributed systems researchers is to find effective ways to exploit these new capabilities to enable

new network services and applications.

The development of such services and applications requires experimentation in realistic testbed settings, using routers that reflect the architecture and performance characteristics of modern commercial systems. At the same time, academic researchers require flexible, open systems, that can be fully understood and that are easily extended to enable new features. Systems provided by commercial vendors are generally of only limited use to networking researchers, precisely because they are not open for modification by the network research community. This paper has described a flexible, open router that can be an effective tool for systems researchers in search of a more suitable experimental platform.

References

[1] Chaney, T., Fingerhut, A., Flucke, M., Turner, J. [1997]. "Design of a Gigabit ATM Switch," *Proc. of INFOCOM 97*, IEEE, Kobe, Japan.

[2] Cisco Systems, Inc. "Tag Switching: Uniting Routing and Switching for Scalable, High-Performance Services," White Paper, http://www.cisco.com/warp/public/cc/cisco/mkt/ios/tag/tech/tagsw_wp.pdf, 1998.

[3] C-port Corporation. "Communication Processors: a Definition and Comparison," http://www.cportcorp.com/solutions/docs/c5wpaper.pdf, 1999.

[4] Decasper, D., Parulkar, G., Choi, S., DeHart, J., Wolf, T., Plattner, B. "A Scalable, High Performance Active Network Node," *IEEE Network*, January/February 1999.

[5] Decasper, D., Plattner, B. [1998]. "DAN - Distributed Code Caching for Active Networks," *Proc. of INFOCOM 98*, IEEE, San Francisco.

[6] Decasper, D., Dittia, Z., Parulkar, G., Plattner, B., "Router Plugins - A Software Architecture for Next Generation Routers," *Proceedings of ACM SIGCOMM'98*, September 1998.

[7] Decasper, Dan. "A Software Architecture for Next Generation Routers," doctoral dissertation, Swiss Federal Institute of Technology in Zurich (ETHZ), 1999. Available through http://www.arl.wustl.edu/ dan.

[8] Dittia, Z., J. R. Cox, Jr., G. Parulkar. "Design of the APIC: A High Performance ATM Host-Network Interface Chip," *Proc. of INFOCOM 95*, IEEE, Boston, MA.

[9] The NetBSD Foundation. http://www.NetBSD.ORG/, 1999.

[10] Shreedhar, M., Varghese, G. [1995]. "Efficient Fair Queueing using Deficit Round Robin," *Proc. of SIGCOMM 95*, ACM, Cambridge, Mass.

[11] Srinivasan, V., Suri, S., Varghese, G. [1999]. "Packet Classification using Tuple Space Search," *Proc. of SIGCOMM 99*, ACM, Cambridge, Mass.

[12] STS Technologies. http://www.ststech.com.

[13] Tenenhouse, D., Smith, J., Sincoskie, W., Wetherall, D., Minden, G. [1997]. "A Survey of Active Network Research," *IEEE Communications*, 35:1, 80-86.

[14] Washington University Applied Research Laboratory. "Gigabit Networking Technology Distribution Program," http://www.arl.wustl.edu/gigabitkits/kits.html.

[15] Wetherall, David and David Tenenhouse. "ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols," *Proceedings of IEEE OpenArch 98*, April 1998.

[16] Wolf, T., D. Decasper, C. Tschudin. "Tags for High Performance Active Networks," submitted to *OPENARCH 2000*, IEEE, Tel-Aviv, Israel, 2000.