P4: A Platform for FPGA Implementation of Protocol Boosters

Ilija Hadžić and Jonathan M. Smith
ihadzic@ee.upenn.edu, jms@cis.upenn.edu

Distributed Systems Laboratory, University of Pennsylvania*

Abstract. Protocol Boosters are functional elements, inserted and deleted from network protocol stacks on an as-needed basis. The Protocol Booster design methodology attempts to improve end-to-end networking performance by adapting protocols to network dynamics.
We describe a new dynamically reconfigurable FPGA based architecture, called the Programmable Protocol Processing Pipeline (P4), which provides a platform for highly-flexible hardware implementations of Protocol Boosters. The prototype P4 is designed to interface to an OC3 (155 Mb/s) ATM link and perform selected boosting functions at this line rate.
The FPGA devices process the data stream as a pipeline of processing elements. Processing elements are downloaded and activated dynamically, based on policies used by the controller to choose configurations. As modules become unnecessary they are removed from the pipeline chain.

1 Introduction

Network protocols are designed to meet application requirements for data communications, including security, reliability and performance. The dominant design and implementation process for protocols begins by enumerating the requirements for the protocol, and then designing a protocol that provides the necessary features end-to-end[SRC81]. Optimization consists of identifying common cases and implementing fast paths for these cases; TCP/IP is a case study[CJRS89]. The resulting protocol is robust end-to-end and typically provides good performance. However, extremely poor performance can result when the assumptions permitting fast path execution are not met.

1.1 Protocol Boosters

"Protocol Boosters"[FMS96] are protocol elements intended to be transparently inserted into and deleted from protocol stacks on an as-needed basis, e.g., in response to unanticipated or changing network characteristics.

* This research was supported by DARPA under Contracts #NCR95-20963 and #DABT63-95-C-0073, the AT&T Foundation, the Hewlett-Packard Corporation, the Intel Corporation and the Altera University Grants Program.

A *policy* associated with the booster is used to selectively insert, delete and invoke the protocol functions. For example, a Forward Error Correction code (FEC) might be used over a wireless data link to bring its error behavior into an acceptable operating range, without using the FEC end-to-end [McA95]. The error performance of the subnet is thus “boosted” to an acceptable level to improve end-to-end performance. Figure 1 shows a booster used in a network, in this case boosting a subnet between an end-host and a router.

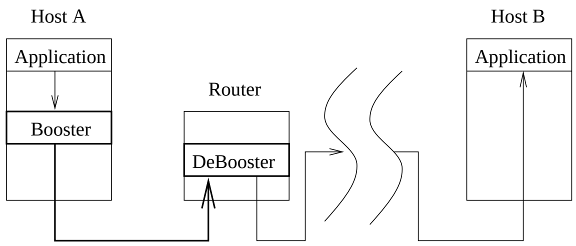

Fig. 1. Boosting a link or subnet

A *transparent* booster does not modify the packet it boosts. For example, an FEC booster may send FEC packets *in addition to* the data packets it encodes. Non-transparent boosters, on the other hand, modify data packets. For example, a compression booster might be used to reduce the bandwidth requirements.

The recent discussion of boosters in Feldmeier, *et al.*[FMS96] restricts boosting to transparent boosters to cope with dynamic routing. Non transparent boosters generally need a static route (e.g., an ATM VC) for successful implementation. Our platform is capable of implementing either type of booster.

1.2 Previous Work

The Protocol Booster idea has been investigated in a software implementation [MCS97], where a FreeBSD TCP/IP protocol stack was boosted with various functions. The major finding was that the operating system overhead to support boosters was minimal; the major cost of boosting was the processing cost of the boosting algorithm(s) applied to the data stream. This suggested that any boosting scheme requiring high performance would need a way to first provide the flexibility required by design methodology, and second, provide this flexibility with high processing performance.

Field-programmable gate array (FPGA) devices offer the degrees of flexibility and performance required for protocol boosting. Our FPGA-based architecture, the Programmable Protocol Processing Pipeline (P4) provides a new platform for Protocol Boosters hardware implementation.

The P4 operates on streams of Asynchronous Transfer Mode (ATM) cells, and serves as a generic platform for running Protocol Boosters over the ATM infrastructure. This paper's primary focus is the P4's logical structure, and describes the architectural solutions used in realization of a P4 prototype. Although the P4 is designed to operate in the ATM environment, the concept is applicable in any other protocol.

The next section, Sect. 2, presents the architecture of the P4, and briefly describes its operation. Section 3 details the functional modules and their operation. Section 4 points out questions associated with the implementation and usage of the P4 architecture and outlines the the longer-term project goals. Section 5 concludes the paper with a summary of what the architecture proves, and next steps in our research.

2 Basic Architecture

Figure 2 shows a block diagram for the P4 architecture. The core of the architecture is a pool of Altera Flex8000 family[Alt96] FPGA devices, acting as processing elements (PEs), and a switching array that provides interconnections among processing elements.

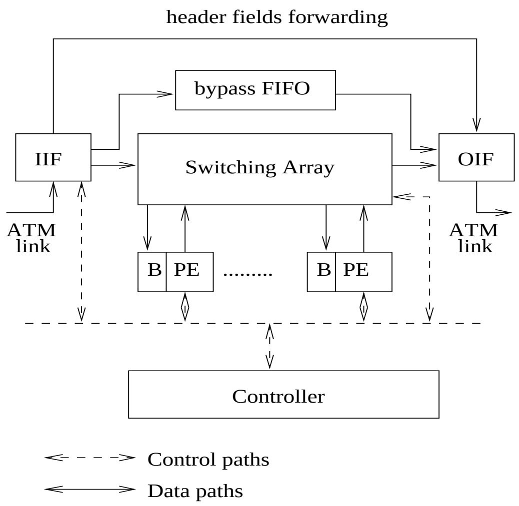

Fig. 2. Block diagram of the P4 Architecture

PEs typically form a pipeline chain via the switching array. Each PE implements one function in the pipeline chain. For example, if the FEC was to be added into the protocol stack, the available FPGA device would be configured to implement the encoder on the transmitter side; on the receiver side, one of the PEs would be configured as the decoder.

Associated with each PE is a FIFO buffer (B). A PE reads the data from its FIFO buffer, performs its processing, and writes into the FIFO buffer associated with the next device in the chain. Connection to the next device is achieved via the switching array.

Data from the ATM link are received in parallel (octet by octet), at 19.44MHz, making the board capable of processing the cell stream in real time at 155Mb/s. In our current experimental setup, we use the HP75000 Broadband Test-system equipped with the Network Impairment Emulation Module[DGH+97]. The P4 board connects to the parallel line interface of the HP75000[Hew94]. The system can generate test traffic patterns and introduce impairments (cell losses, bit errors etc.) and thus emulate the conditions encountered in real networks.

While the current version of the P4 board does not have a Sonet/SDH interface, timings of the parallel interface are compatible with the timings of commercially available framing devices[Tex95], making future addition of such a Sonet/SDH interface straightforward. At this stage, we are primarily interested in the implementation of the P4 functionality and the role it can play in protocol processing.

3 Functional Elements

3.1 Input Interface (IIF)

The input interface (IIF) of Fig.2 provides initial processing of the cells. It is realized by a specifically configured FPGA device downloaded on powerup. The IIF is the first device in the chain and it does not have a FIFO buffer preceding it. It connects to the parallel cell stream which may come either from the test equipment or a Sonet/SDH framing device.

The IIF scans the input stream for cells belonging to the processed virtual circuit and stores them in the FIFO buffer associated with the first PE in the pipeline chain. All other cells are stored in the bypass buffer together with the header fields. Thus the IIF selects the virtual circuit to boost assuming that the VPI/VCI corresponds to a single application (a realistic assumption for an ATM network).

Headers of cells on the boosted channel are stripped off and payload data are stored into the FIFO buffer. Header information is passed directly to the output interface (OIF) via a special data path, bypassing the PEs.

There are two reasons for stripping off the header fields:

1. Having processing elements deal with only the payload reduces their complexity, saving space in the FPGA device and enabling us to put more *processing* functionality in a PE.
2. Some processing algorithms may change the size of the actual data (e.g. data compression). In such cases it is clearly simpler to remove the header before the processing and reconstruct cells at the output.

3.2 Output Interface (OIF)

The output interface (OIF) of Fig.2 complements the IIF. The device reads from the FIFO buffer associated with it, creating a boosted cell stream. If the buffer is empty, the OIF will try to read the cell from the bypass FIFO. If both buffers are empty, the OIF will not generate a cell. The IIF and OIF operation is illustrated in Fig.3.

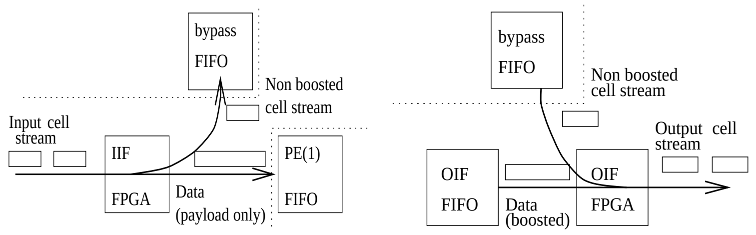

Fig. 3. The IIF/OIF operation

The OIF can be delayed by any processing element in the chain. For example, if one of the processing elements performs data compression, it would typically request a delay in cell generation if it produced output shorter than 48 bytes before new data appeared. Delaying the cell for a limited amount of time might avoid padding the payload with idle data and improve the bandwidth utilization. However, a cell must not be arbitrarily delayed; we will study tradeoffs across a selection of traffic types.

3.3 Switching Array

The *switching array* improves flexibility of the P4 architecture as functions in the PEs are not necessarely commutative. If a new processing element must be activated, it can be downloaded into any available FPGA device. The switching array will allow placing of the processing element at the appropriate point in the pipeline chain regardless of the actual FPGA device used; it serves to virtualize pipeline ordering.

3.4 Processing Element (PE)

The Processing Element (PE) is the generic part of the architecture. It is a combination of a FIFO buffer and an FPGA device. The output of the FPGA device is the set of control and data signals compatible with the FIFO buffer input. Processing consists of reading from the FIFO buffer, processing the data and writing it into the FIFO buffer associated with the next PE in the chain.

In the minimal case (no PE active), the IIF reads the data from the link and writes into the FIFO buffer of the OIF. The boundary between PEs is the FIFO data and control bus interface. Since this is well defined for the given FIFO[Cyp95] it is simple to insert and remove PEs in the P4's chain.

3.5 Controller

From the controller's perspective, PEs are I/O devices. The controller has a pool of FPGA configurations with which it can implement a processing algorithm. Depending on the policy and network conditions, it chooses the appropriate set of configurations and downloads the FPGA devices on demand. Downloading is managed by the controller, and the FPGA is in the passive configuration mode[Alt94]. Using the switching array, the controller creates the chain with processing elements ordered appropriately.

3.6 P4 Role and Placement in a Network

The P4 board is intended to work as an "edge" device for the boosted portion of a network cloud as shown in Fig.4. Although the prototype is designed to operate on the stream of ATM cells, the same concept can be applied to any type of packet stream.

The P4 can work as a standalone unit with its own controller or as part of a switch or other network element. In the latter case the P4 can be an integral part of the switch and thus managed by the network element's controller. A single controller may be responsible for multiple P4 boards.

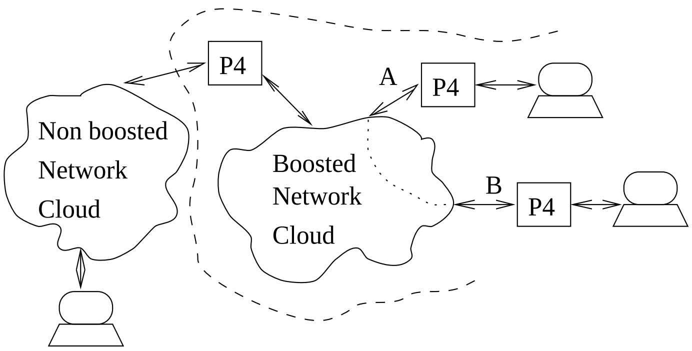

Fig. 4. P4 as the part of the network

In Fig.4, the boosted portion of the network has a P4 board associated with all access points. If a user connected to the P4 enhanced access point communicates with another user connected to the P4 enhanced access point, their communication can be point-to-point boosted provided that there is a route entirely through boosted portion of the network. An important property of the P4 as used in this mode is that if the route has to pass through the non-boosted portion of the network, the P4 board at the boundary will act as a conversion unit by deboosting the outgoing traffic and boosting the incoming traffic. This enhances the interoperability of the scheme.

4 Research Questions

The P4 board will aid exploration of the design space for hardware implementation of Protocol Boosters. It should process the 155Mb/s ATM cell stream in real time, and allow dynamic reconfiguration with minimal delay. This section describes the performance metrics for the architecture.

4.1 Protocol Processing Performance

One goal of Protocol Boosting is end-to-end performance improvement via adding a parasitic module to the original protocol[FMS96]. As shown with the software prototype of Protocol Boosters[MCS97], some functions can be efficiently implemented in software resulting in better end-to-end performance. However, functions such as data encryption significantly degrade throughput while adding their functionality (e.g., privacy of communication).

Preliminary measurements with the P4 board using the convolutional FEC booster have shown that the processing delay is minimal compared to the cell transmission delay. Due to the pipelined nature of the architecture, link throughput is preserved[2] and the processing overhead is reflected in latency through the P4. The limit on the pipeline depth depends on the existing propagation delay and the total delay acceptable by the application.

4.2 Resource Management Issues

A second important issue is the functional "agility" of the P4. By agility we mean latency for reaction to the network anomalies. For example, consider the case where we want to address load-induced congestion by adding data compression to the link protocol (provided we can successfully compress!). At a link speed of 155Mb/s the time to download the FLEX8K device[Alt96] can be too long to successfully address the problem of congestion.

For some applications, agility is not a critical issue. For example, a user is not likely to notice the slight delay in adding FEC to audio traffic to improve sound quality. Thus, there are applications which require very fast reaction ("agility") from the protocol booster and there are also applications where the low agility can be tolerated.

Use of the P4 architecture for non-trivial Protocol Boosters implementations raises the question of management of limited resources (namely FPGA devices which implement the processing elements) to achieve statistically (i.e., almost always) high agility. Resource management in the P4 architecture is analogous to cache management in a general purpose computer system. A cache controller takes advantage of the correlation between accesses to nearby memory locations and statistically reduces the memory access time. In a similar fashion, we can

[2] It should be understood that if the booster generates additional cells, the input data can not arrive at the maximum bit rate. However this limitation is not due to the P4 processing overhead but the link capacity.

achieve statistically improved agility of the Protocol Booster running on the P4 architecture.

A well-designed management algorithm will try to keep the agility sensitive configurations pre-loaded in the FPGA devices and clear them only if deemed necessary (i.e., if an FPGA device is needed for some other algorithm and there is no free device available). Resource management algorithms and statistical properties of the events in the network that can be used to trigger the activation of the specific Protocol Booster will be major (and we believe generalizable) results of this research.

4.3 Policy Modules

An important part of the Protocol Booster is the policy module. The policy module decides when a function should be activated based on its observations of the data stream.

In this paper, we have focused on the issues of implementing processing functions (i.e., mechanism modules) in the P4 architecture. As the controller decides which function to download, enforcing the policy is the task of the controller.

The controller can delegate some of this role to a PE by configuring it to monitor the link instead of performing the actual processing. The controller has full access to the PE and can communicate with it via its data bus, and thus some PEs can be programmed to monitor the events on the link and provide the controller with information crucial to policy decisions.

Thus, the P4 controller *enforces* the policy but PEs can perform functions which are part of both policy and mechanism module.

4.4 Need for Synchronization

Boosting a portion of the network means adding some processing on one side and restoring the original data on the other side. For convenience, assume a non-transparent booster and call these functions boosting and deboosting respectively. The booster needs to be synchronized with the debooster[3], that is, the debooster must know when to engage. This requires a signalling protocol, either explicit or implicit.

Consider the infrastructure shown in Fig.4, and suppose that side A decides to load a booster and that its matching debooster must be loaded on side B. Side A needs a method of in-band signalling to inform side B when the booster is to be activated. As a simple scheme for testing the P4 architecture prototype, we will use in-band signalling with control cells in the same virtual channel as the data, and use the payload-type field[ATM94] to distinguish signalling and user data.

[3] This is not strictly required for transparent boosters[FMS96], but of course they are more effective when synchronized!

4.5 Extension to Multiple Virtual Circuits

The architecture has been described assuming single virtual channel (VC) processing. The current prototype is limited to one VC, as the focus is on the basic P4 architecture. In later versions we plan to implement multiple VC processing on one board.

Besides the pipeline topology, the switching array can connect processing elements in the parallel pipelines and provide different processing on different VCs. It is also straightforward to modify the IIF to use VPI only instead of the VPI/VCI combination to identify the boosted cell stream.

An interesting approach (suggested by Bill Marcus of Bellcore) is to use "context switch"-like processing in the PE. The state of the PE can be stored away in the content addressable memory (CAM) and VPI/VCI can be used as an index to the state information in the CAM. A PE can thus switch state accordingly and apply the same algorithm to multiple concurrent virtual channels.

5 Conclusion

We have presented an architecture which has the software-like flexibility required for the Protocol Boosters design methodology, while providing the performance expected from a hardware implementation. However, the P4 is more generally useful; for example, as an array of downloadable devices which perform protocol processing, it might function as a high-performance element in an *Active Network*[SDG+96].

The P4 prototype raises several issues in networking subsystem design. On one hand, it is a generic architecture with a limited number of resources (i.e., the processing elements) which need a good management policy. On the other hand it is a platform which allows the implementation of certain functions in hardware and gives us a new tool for addressing the balance between hardware and software in protocol implementation. It is particularly important for evaluating these engineering tradeoffs that the generic structure of the P4 allows fine-grained control of the boundary between functions implemented in hardware and software.

The prototype is operating as a wirewrap card. Next is PCB implementation, and evaluation of various processing functions that will run on the P4 board. The construction of the testbed involving the ATM subnet enhanced with the P4 boards at the entry points will allow us to outline the domain of applicability of P4-like architectures for protocol processing, and evaluate algorithms for dynamically allocating the hardware resources based on application demands and network dynamics.

6 Acknowledgements

Bill Marcus of Bellcore provided a number of helpful comments in addition to the multiple VC scheme (Sect. 4.5). Tony McAuley, also of Bellcore, helped clarify

our thinking on the roles of transparency, routing and booster state. Dave Feldmeier of MUSIC Semiconductors made some useful suggestions. Tyler Arnold has been investigating the implementation of a convolutional FEC booster for use in the P4.

References

[Alt94] Altera, Corporation, 2610 Orchard Pkwy., San Jose, CA 95134. *Configuring the FLEX8000 devices - Application Note 33*, 3 edition, May 1994.

[Alt96] Altera, Corporation, 2610 Orchard Pkwy., San Jose, CA 95134. *FLEX8000 Programmable Logic Device Family - Data Sheet*, 8 edition, June 1996.

[ATM94] ATM Forum. *ATM User Network Interface Specification, version 3.1*, September 1994.

[CJRS89] David D. Clark, Van Jacobson, John Romkey, and Howard Salwen. An analysis of tcp processing overhead. *IEEE Communications Magazine*, 27(6):23–29, June 1989.

[Cyp95] Cypress Semiconductor Corporation, 3901 N. 1st St, San Jose, CA 95134. *CY7C421 Data Sheet*, January 1995.

[DGH+97] R.W. Dmitroca, S.G. Gibson, T. R. Hill, L. M. Morales, and C. T. Ong. Emulating atm network impairments in the laboratory. *Hewlett-Packard Journal*, 48(2):45–50, April 1997.

[FMS96] D. C. Feldmeier, A. J. McAuley, and J. M. Smith. Protocol boosters. *submitted to IEEE JSAC Special Issue on Protocol Architectures for the 21st Century*, 1996. U. Penn CIS TR MS-CIS-96-34.

[Hew94] Hewlett - Packard, IDACOM Telecommunications Operation, 4211 95 Street, Edmonton, Alberta, Canada. *Optical Line Interface User's Guide*, 3 edition, June 1994.

[McA95] A. J. McAuley. Error control for messaging applications in a wireless environment. In *INFOCOM 95*, April 1995.

[MCS97] A. Mallet, J. D. Chung, and J. M. Smith. Operating systems support for protocol boosters. In *Proceedings, HIPPARCH Workshop*, June 1997.

[SDG+96] J. M. Smith, D.J.Farber, C. A. Gunter, S. M. Nettles, D. C. Feldmeier, and W.D. Sincoskie. Switchware: Accelerating network evolution (white paper). Technical report, University of Pennsylvania, URL: http//www.cis.upenn.edu/ jms/white-paper.ps, June 1996.

[SRC81] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. In *Proceedings of the 2'nd IEEE International Conference on Distributed Computing Systems*, pages 509–512, April 1981.

[Tex95] Texas Instruments, P.O. Box 655303, Dallas, TX 75265. *TNETA 1500, 155.52Mb/s Sonet/SDH ATM receiver/transmitter*, July 1995.

This article was processed using the LaTeX macro package with LLNCS style