On-the-fly Programmable Hardware for Networks

Ilija Hadzic and Jonathan M. Smith
Distributed Systems Laboratory, University of Pennsylvania*
ihadzic@ee.upenn.edu, jms@cis.upenn.edu

Abstract

Ongoing research in adaptive protocols and active networks has presumed that flexibility is offered exclusively through software systems, and the performance implications have generated considerable skepticism. The Programmable Protocol Processing Pipeline (P4) exploits the dynamic reconfigurability of RAM based Field Programmable Gate Arrays (FPGAs) to provide both hardware performance and dynamic functionality to network components.

We use forward error correction (FEC) as an example of a protocol processing function. Our measurements show that the P4 is able to process the data stream at OC-3 (155 Mbps) link rate, and consequently improve TCP performance in noisy environments.

1 Introduction

A desire for flexible network infrastructures has stimulated research into adaptive protocols and active networks. This research[14] has presumed that flexibility is offered exclusively through software systems, and the performance implications have generated considerable skepticism. In particular, a number of researchers[9] have proposed that programmability be restricted to the control plane, as they believe that high data throughput cannot be achieved concurrently with dynamically interposed functions.

However, flexibility is not exclusive to software systems: new programmable logic devices can be reprogrammed rapidly enough so that network components can operate at hardware speeds while providing dynamic functionality. The growth in size and speed of state of the art programmable logic devices has stimulated new fields of research, *e.g.,* reconfigurable computing[15].

We are exploring the application of dynamically reconfigurable hardware to adaptive protocols and active networks. To explore the design space where high speed requirements make software implementation a bottleneck, we have constructed an FPGA-based architecture called the Programmable Protocol Processing Pipeline (P4)[7]. We thus achieve functional acceleration with special purpose hardware while maintaining a software-like flexibility of the system.

We focus on the example of TCP/IP performance in a noisy environment. We protect against noise-induced errors with the FEC, and demonstrate the convolutional encoder and Viterbi decoder operating at OC-3 (155Mbps) data rates on the P4. The next section briefly describes the P4 architecture.

*This research was supported DARPA under Contracts #NCR95-20963 and #DABT63-95-C-0073. Additional support was provided by the AT&T Foundation, the Hewlett-Packard Corporation, the Intel Corporation and the Altera University Grants Program. This paper has been submitted to IEEE GLOBECOM 98 for review.

2 P4 Architecture

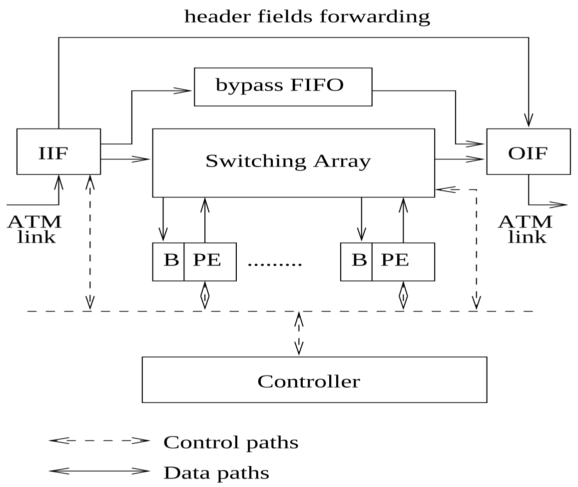

Figure 1: P4 Architecture

The architecture of the P4 is shown in Figure 1. It composes a set of RAM based FPGA devices (Altera FLEX8000[1]) in a pipeline, with a switching array selecting which devices are engaged in processing a data stream. FPGA devices allow implementing protocol processing algorithms in hardware, while providing dynamic functionality through the run time reconfiguration.

Processing elements in the P4 are organized into a pipeline of programmable logic devices interconnected by the switching array. Each device has a FIFO buffer associated with it. A processing element 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. The switching array can dynamically include or exclude processing elements, or reorder them on an as-needed basis.

When needed, a protocol processing function (in the form of an FPGA configuration) is added by downloading a free device, and inserting this device into the pipeline chain. Unnecessary functions are switched out of the processing chain and the device becomes free. Altera's Flex 8000 devices require about 100ms to be reloaded, but can be switched in and out of the data path within a microsecond. The gate arrays can thus be viewed as a cache for selected protocol processing functions.

The P4 prototype uses ATM cells as a convenient unit of processing. While the architecture is not ATM-specific, use of ATM allows interoperation with existing systems and validation of performance in 100+ Mbps operating regimes.

3 FEC Booster

The P4 prototype has been constructed as part of the Protocol Boosters project[6], which takes the approach of dynamically adding and deleting protocol functions. The P4 illuminates a design subspace where high speed requirements force the implementation of certain functions in hardware.

We have chosen an FEC as an example protocol processing function which might operate on an as-needed basis for greater efficiency. A convolutional encoder and Viterbi decoder were implemented to allow experimental evaluation. Our goal was *not* to construct a highly optimized code for a given link, but rather to explore the feasibility of performing a complex protocol processing function using the limited set of resources offered by the P4. Thus, the FEC was optimized for implementation on the P4 and operation at the P4's OC-3 data rate.

3.1 Implementation

Bits of each data octet are grouped in four chunks of two bits and encoded independently using four parallel, rate 1/2, constraint length 3, convolutional encoders. Each encoder accepts two bits from the current octet and produces four output bits. Four parallel encoders thus produce two octets of data which are clocked out at 19.44 MHz, resulting in the output bit rate of 155Mb/s.

On the decoder side, four parallel decoders operate independently on the groups of two bits producing one bit of original data stream. Figure 2 illustrates this procedure. Input data is clocked in at 19.44MHz, so the input bit rate is 155Mb/s. Due to the complexity of the Viterbi decoder, each decoder had to be located on a separate processing element. The first processing element (Decoder 1 in Figure 2) takes two bits of the encoded octet and produces one output bit. The other six bits are passed to the next processing element unmodified. So the output octet of the first processing element consists of six original encoded bits, one decoded bit and one unused bit. At the output of the fourth decoder each octet contains four decoded bits and four unused bits. Finally, two such octets are combined to reconstruct the original data.

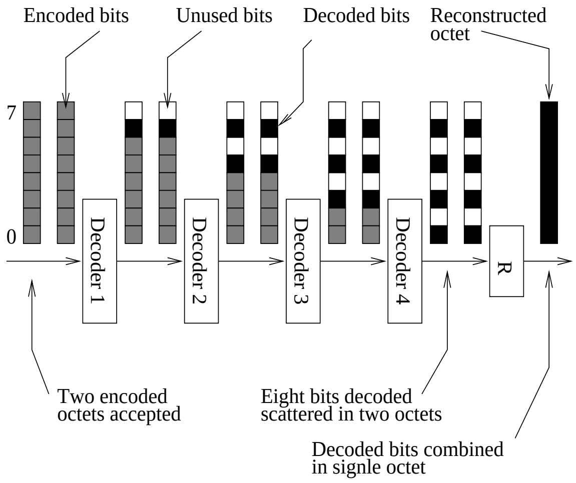

Figure 2: Decoding process on P4

3.2 Robustness

An important issue in protocol design is robustness. Although it protects user data from bit errors, convolutional encoding may increase the risk of other impairments such as cell losses and cell misinsertions if no countermeasures are applied. In general, the output of the Viterbi decoder depends on the history of its inputs. If a cell is lost, missing data may cause unpredictable behavior, and the error can propagate far into the future. To improve robustness in such cases, the encoder resets its state every 24 bytes (half the ATM cell) and the decoder resets its state after every cell.

The encoder generates two cells for each input cell. Both cells have the same value for the user indication bit in the ATM header. If the encoded cell within the AAL-5 protocol data unit (PDU) is lost, there will be a mismatch in the user indication bit at the end of the AAL-5 PDU. Prior to decoding, data are passed through the front end processing unit which checks for matching user indication bits. Only pairs of cells that match are passed for further decoding. If a mismatch is found, an all-zero cell with the appropriate user indication bit will be inserted as shown in Figure 3. This will, of course, result in a series of bit errors after decoding, but will prevent any error propagation that might otherwise result. Only the AAL-5 PDU whose cell has been lost will be affected. It can be easily verified that the front end processing unit will also successfully isolate bad AAL-5 PDUs in the case of cell misinsertion, thus avoiding error propagation.

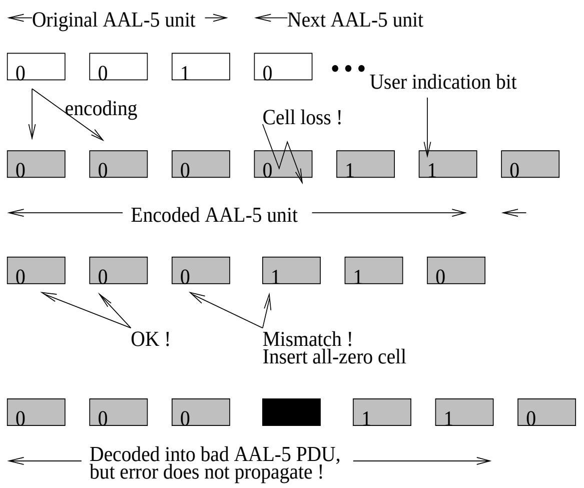

Figure 3: Protection from error propagation due to cell loss

4 Application

An important motivation for the Protocol Boosters concept is the problem encountered when protocols optimized for certain conditions operate outside those conditions; they perform extremely poorly. Flexible adaptive protocols and active networks cope with this problem by dynamically adapting the protocol stack to one appropriate for the current conditions in the network.

Wireless ATM[11] is an example where the protocol requires modification, as the original assumptions for ATM link reliability are no longer met. In an effort to improve the link quality, modifications of the link layer that incorporate strong FEC in combination with ARQ have been suggested[4, 10, 3, 16]. It is, of course, unlikely that an optimal error control scheme meeting the needs of all applications under all possible conditions exists. In [12] the author of the NEC Wireless ATM prototype[5] has pointed out that each service type will require an appropriate error control scheme, implying that the error control is not a static mechanism. [10] considered protecting only the header of the ATM cell to prevent extensive cell losses and misroutings, and leaving the protection of the payload to the higher layers depending on the desired quality of service.

In addition to the different error control schemes needed for different service types, the bit error rate on a wireless link is changing over time.

For adaptive protocols, FEC can be viewed as the functional element of the protocol stack which can be added, removed or changed on an as needed basis. With the spectrum of FEC implementations of varying strengths and complexities available, dynamic protocol can select the implementation that best fits the current conditions and QoS requirements. In the enhanced network infrastructure provided by the P4, different FEC implementations are available as FPGA configurations. When an appropriate coding scheme is selected, processing elements in the P4 are configured and the result is P4 operating as specialized hardware in the network. If the FEC algorithm must be replaced, the processing element is reconfigured and new specialized hardware is activated, reusing the same physical device.

5 Experiments

Our experimental work evaluates the effect on link throughput of the FEC implemented on the P4. With a tunable bit error rate induced on the link, we measured the TCP throughput seen by the application with and without the FEC booster described in Section 3.

5.1 Test Setup

The experimental setup is shown in Figure 4.

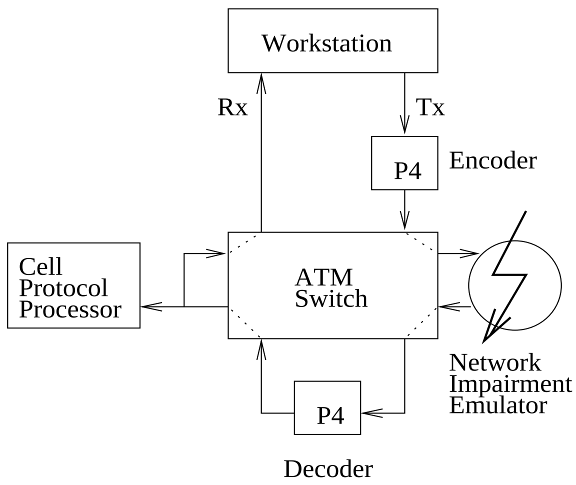

Figure 4: Experimental setup used in testing the FEC booster

The host is an Intel Pentium PC running Linux kernel, release 2.0.29, with the “ATM on Linux”[2] patch and a Fore Systems PCA200E ATM adaptor. Throughput testing is done with `ttcp`. For convenience, we used single test machine with source and sink running as two separate processes. Since we were interested in testing the impact of the P4 on TCP throughput and not the impact of the workstation, this setup can deliver useful results.

Cells transmitted by the workstation are encoded using the first P4 in the test setup. At the output of the first P4, the utilized bandwidth is twice the bandwidth generated by the workstation due to the additional cells. To prevent buffer overflows in the operating P4, the device driver in the workstation must be rate controlled. Our rate limiting mechanism forces an idle period between the transmission of two consecutive packets so that the encoder in P4 has an opportunity to insert all generated packets. There are tradeoffs among the buffer size on the P4, the maximum segment size for IP running over the link, and the length of the enforced idle period.

Encoded cells are passed through a noisy link, emulated by inserting bit errors with the Network Impairment Emulator[13]. We vary the bit error rate and measure the TCP throughput seen by the receiving process on the workstation with and without the FEC booster in place.

The second P4 board decodes the cells and corrects any correctable bit errors. Decoded data are passed through the Cell Protocol Processor[8] which acts as the passive monitoring device. We use the Cell Protocol Processor to monitor the link traffic and the error rate after decoding.

5.2 Results

We ran the `ttcp` throughput tests for four cases: (1) without P4 boards in the data path; (2) with P4 boards doing no processing; (3) with P4 boards doing no processing, rate control on; and (4) with P4 boards configured as FEC encoder and decoder.

We varied the bit error rate (BER) from 10^{-12} to 10^{-4} with an exponentially distributed time between two consecutive bit errors (*i.e.,* a Poisson error distribution). Results from the first case provide a baseline measurement. In the second case we tested if inactive P4 hardware had any impact on the results. The first two cases exhibit almost identical results: an enormous dropoff in TCP/IP performance (the throughput is on a logarithmic scale in Figure 5). This is due to TCP/IP's strategy in the face of packet loss, which is to assume that the loss was a result of congestion rather than noise. The result is that the TCP/IP congestion window is rapidly reduced to the point where the protocol becomes "stop-and-wait", with the consequences shown in Figure 5. The third case shows the effect of rate control, namely that the throughput starts off considerably lower (a factor of 4 less) but drops off as rapidly as the first two cases in the face of error. The reason this test was performed was to separate the costs of rate control from the costs associated with the FEC processing.

In the last experiment, we measured TCP throughput with the FEC in place, and rate controlled, as before. As expected for the low BER region, the FEC booster does additional processing and uses extra bandwidth for the redundancy, beyond the cost of rate control, taking its throughput to about 8 Mbps. In the high BER region, the TCP protocol stack benefits from FEC in reducing the number of retransmissions and keeping the value of TCP window size larger. Without FEC, TCP completely stalls at BER below 10^{-4}, while it is still able to operate with FEC in place.

Figure 5 shows the logarithmic plot of the mean value of measured throughput as a function of BER and Table 1 shows 90% confidence intervals for measured throughput. Log plots are used since BERs of interest covers many orders of magnitude. The upper solid line presents the throughput without the P4 in the datapath; the dashed line following it is the throughput with an idle P4. The overhead introduced by the P4 hardware is negligible. The lower solid line shows the TCP throughput with rate control and an inactive P4. Finally, the dashed & dotted line shows the measured throughput with P4 running the FEC booster. For BERs greater than 10^{-6}, TCP gains from FEC. Given our earlier explanation of TCP's response to packet loss, it should be clear that the FEC, in reducing the impact of noise, reduces the probability of the incorrect assumption of congestion. Thus, the performance is improved.

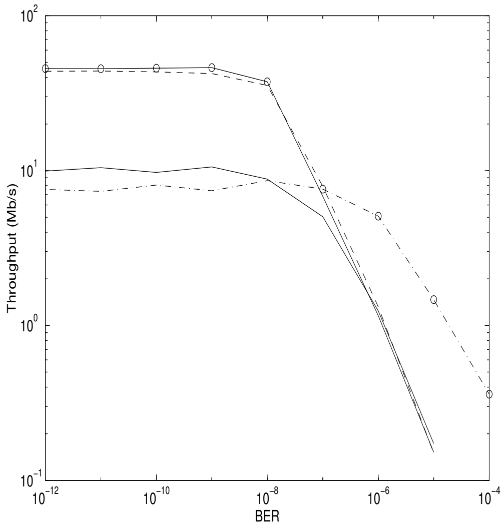

Figure 5: Mean value of measured throughput

The graph in Figure 5 illustrates an opportunity for an adaptive protocol. In particular, the intersecting curves at a BER of ca. 10^{-7}, suggest that FEC be employed only when the BER exceeds 10^{-7}. Thus, a

BER	without P4	with P4, no FEC	no FEC, choking	with P4, with FEC
0	$[45.46, 46.09]$	$[43.80, 43.95]$	$[10.36, 11.50]$	$[6.96, 8.90]$
10^{-12}	$[45.23, 45.85]$	$[43.68, 43.86]$	$[9.43, 10.42]$	$[7.15, 7.98]$
10^{-11}	$[45.11, 45.90]$	$[43.83, 44.06]$	$[9.96, 10.93]$	$[6.57, 8.14]$
10^{-10}	$[45.59, 46.18]$	$[42.59, 44.18]$	$[9.30, 10.16]$	$[7.26, 8.87]$
10^{-9}	$[45.87, 46.55]$	$[41.19, 43.51]$	$[9.98, 11.16]$	$[6.59, 8.24]$
10^{-8}	$[34.50, 40.40]$	$[33.02, 37.94]$	$[8.07, 9.54]$	$[7.82, 9.42]$
10^{-7}	$[6.48, 7.15]$	$[6.72, 9.11]$	$[4.63, 5.43]$	$[6.95, 8.22]$
10^{-6}	$[1.12, 1.19]$	$[1.27, 1.36]$	$[1.21, 1.27]$	$[4.71, 5.44]$
10^{-5}	$[0.15, 0.16]$	$[0.145, 0.155]$	$[0.17, 0.18]$	$[1.43, 1.51]$
10^{-4}	N/A	N/A	N/A	$[0.35, 0.37]$

Table 1: Confidence intervals for measured throughputs

protocol booster's policy module would constantly monitor the conditions on the link (*e.g.,* using AAL-5 CRC or IP checksums), and switch on the FEC as needed. In Figure 5, the line followed by an ideal adaptive protocol is marked by an "O". In the Protocol Boosters framework, the FEC processing is mechanism, under control of the aforementioned "Policy".

6 Generalizing Adaptive FEC in Hardware

The P4 demonstrates near-software flexibility and performance comparable to special purpose hardware. We used the example of a convolutional code for FEC which is used as a mechanism for an FEC Booster. Two generalizations can be drawn from this example. First, the mechanism is neither limited to a particular coding rate nor a single coding algorithm. Sets of codes appropriate for different BERs or burst lengths can be made available, and the best code for current conditions in the network can be selected by a policy. If conditions change, the code can be adapted at run time by reconfiguring the P4. Second, the P4 architecture can be used for much more than FEC. Any functional element (*e.g.*, encryption, data compression, traffic shaping modules, *etc.*) implementable on P4 can be used as a protocol booster and added to a protocol stack on an as-needed basis.

7 Next Steps

We are developing a policy module for the FEC boosters, and designing the signalling protocol which would enable P4 boards distributed over the network to synchronize their activities. Our design will have the policy module as part of the controller which manages the P4 board and configures its processing elements. For example, to insert the FEC booster, the following mechanisms are necessary:

- determine the type of the code to be used
- decide when to activate the booster
- signal to the other end to prepare the appropriate decoder and if necessary to the host (as in the case where the host needs to activate the choking mechanism inside the device driver)
- download P4 processing elements on both ends
- activate the encoder and signal to the other end to activate the decoder

Selecting the appropriate booster and when to activate it are the two central roles of the policy module. In a realization, the policy module is a combination of the software running on the controller and the configured hardware running on the P4 assigned to monitoring the conditions on the link and collecting the information necessary for policy decisions.

We intend to design a second generation of the P4. The main limitation of the current version is the lack of buffering for local processing. There are also some small dependencies on ATM. Local buffering is essential in supporting *transparent* boosters, which do not modify the original packet. An example of a transparent booster is an FEC booster that sends the FEC packets *in addition to* the original packets. The price paid here is in the memory resources where the FEC packet is stored during its construction. Due to lack of local memory resources on the P4, implementation of transparent boosters is limited. In the second generation of P4, we are also planning to move processing to a higher level of object than the ATM cell, and process data blocks independently of the underlying protocol.

8 Conclusion

Our goal with the Programmable Protocol Processing Pipeline (P4) architecture was a demonstration that modern hardware allowed a novel investigation of the design space of programmable network infrastructures. In particular, the P4 demonstrated flexibility by loading an FEC into its pool of FPGAs, and this flexibility was employed in end-to-end throughput tests using TCP on an ATM-attached workstation. The TCP results showed that the FPGA-resident code allowed TCP performance in a BER regime where the protocol was previously inoperable.

The performance tradeoffs of the system with and without the FEC suggest the use of a hybrid strategy, using the FEC as-needed, a scheme to which to P4 is well-suited. We believe that among the uses for such a scheme are wireless ATM applications.

The important result of this demonstration is that schemes such as Protocol Boosters and Active Networking for flexible network infrastructures are not limited to poor performance regimes. For functions which can be implemented within the area limitations of FPGAs at any point in time, hardware performance levels can be achieved. Thus, this refutes much of the skepticism which exists in the networking community about the performance of these approaches, and hence their impact on real networks.

9 Acknowledgements

We would especially like to thank to Bill Marcus of Bellcore for providing us with many useful suggestions during the course of this work and for making the Bellcore OC-3 interface boards available for the P4 project.

References

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

[2] Atm on linux home page and source code. URL `http://lrcwww.epfl.ch/linux-atm/`.

[3] M. Barton. Unequal error protection for wireless atm applications. In *IEEE Globecom 1996, Proceedings*, volume 3, pages 1911–1919, November 1996.

[4] J. B. Cain and N. McGregor. A recommended error control architecture for atm networks with wireless links. *IEEE JSAC*, 15(1):16–28, January 1997.

[5] D. Raychaudhur, et. al. Watmnet: A prototype wireless atm system for multimedia personal communication. *IEEE JSAC*, 1(15):83–95, January 1997.

[6] D. C. Feldmeier, A. J. McAuley, J. M. Smith, D. Bakin, W. S.Marcus, and T. Raleigh. Protocol boosters. *to appear in IEEE JSAC Special Issue on Protocol Architectures for the 21st Century*, 1998. also U. Penn CIS TR MS-CIS-96-34.

[7] I. Hadzic and J. M. Smith. P4: A platform for fpga implementation of protocol boosters. In *Field-programmable logic and applications: 7th International Workshop, FPL'97, Proceedings*, LNCS, 1304, pages 438–447. Springer, September 1997.

[8] Hewlett Packard Company, IDACOM Telecommunications Operation, 11120-178 Street, Edmonton, Alberta, Canada T5S 1P2. *AAL Users Guide*, 3 edition, August 1994.

[9] A. A. Lazar, K.-S. Lim, and F. Marconcini. Realizing a foundation for programmability of atm networks with the binding architecture. *IEEE JSAC*, pages 1214–1227, September 1996.

[10] Y. Nakayama and S. Aikawa. Cell discard and tdma synchronization using fec in wireless atm systems. *IEEE JSAC*, 1(15):29–34, January 1997.

[11] K. Pahlavan, A. Zahedi, and P. Krishnamurthy. Wideband local access wireless lan and wireless atm. *IEEE Communications Magazine*, pages 34–40, November 1997.

[12] D. Raychaudhuri. Atm based transport architecture for multiservice wireless personal communications. In *IEEE ICC 1994, Proceedings*, volume 1, May 1994.

[13] R.W. Dmitroca, et. al. Emulating atm network impairments in the laboratory. *Hewlett-Packard Journal*, 48(2):45–50, April 1997.

[14] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden. A survey of active network research. *IEEE Communications*, 35(1):80–86, January 1997. *Earlier version MIT LCS TR #MIT/LCS/TM-557, 1996.*

[15] W. H. Mangione-Smith, et. al. Seeking solutions in configurable computing. *IEEE Computer Magazine*, pages 38–43, December 1997.

[16] M. Yoshida. High-quality subchannel for wireless atm transmission. In *IEEE Globecom 1996, Proceedings*, volume 3, pages 1892–1896, November 1996.