Offloading Multimedia Proxies using Network Processors

Øyvind Hvamstad[1], Carsten Griwodz[2,3] and Pål Halvorsen[2,3]

[1]FAST ASA, Norway [2]IFI, University of Oslo, Norway [3]Simula Research Lab., Norway
e-mail: hvaoyv@fast.no, {griff, paalh}@ifi.uio.no

Abstract

In this paper, we present a system that aims at offloading multimedia proxies using network processing technology for applications like media-on-demand and distributed on-line games. In particular, we have designed, implemented and evaluated a proof-of-concept prototype on the Intel IXP1200 network processor. Our results show that the prototype succeeds in offloading the host machine as no data packets have to be processed by the host CPU, and the prototype is able to perform application layer forwarding using only a fraction of the cycles compared to a traditional architecture, where all packets are processed by the host CPU.

Keywords

Proxy server, IXP network processors, workload offloading

1 Introduction

The increasing availability of low-cost bandwidth for regular users and a large improvement in machine hardware enable new applications, like media-on-demand and distributed on-line games. These applications have different requirements than traditional applications, in particular the timely delivery of data. In such a multimedia scenario, like in many others, a proxy cache aims at reducing the latency, network load and server load, at the prive of an increased latency for access to uncached content. Such a proxy cache may serve many concurrent clients, and in addition to traditional proxy cache operations like serving clients from the proxy cache, forwarding data, making a cache copy of a data element, etc., these intermediate nodes may experience high processing loads due to transcoding, packet filtering, stream aggregation, protocol translation or other application-specific processing.

In this paper, we propose to offload a multimedia proxy cache by using network processing technology and performing both networking and application level operations on-board. As a first step, we have designed a proxy cache for the Intel *Internet Exchange Processors* (IXP) network processor (Intel Corporation, 2001). Our proxy should be able to perform fast low-level data forwarding for urgent low-latency packets, efficient caching operations to reduce resource consumption and application layer multicast to enable multiple receivers of a packet. In our proof-of-concept prototype, we have implemented and evaluated a small subset of the required operations, i.e., a simple RTSP control/signaling server and an RTP forwarding unit. Furthermore, we show that the prototype successfully offloads the proxy host in the data-plane, i.e., no data packets are processed by the host computer during a data forwarding operation, leaving it free to perform other CPU intensive tasks. Our experiments show that the prototype is able to do application layer forwarding using only a small fraction of the cycles compared to a traditional architecture, where all packets are processed by the host CPU.

The rest of this paper is organized as follows: section 2 and 3 give some background on related work and the IXP network processor, respectively. Section 4 describes our ideas and the design of the network processor-based proxy architecture. In section 5, we present the prototype implementation, and a performance evaluation is given in section 6. Finally, we give a conclusion and directions for future work in section 7.

2 Related Work

On-board network processing units have existed for some time with the initial goal of moving the networking operations that account for the most CPU time from software to hardware. As the systems have improved, the fourth generation network processors aim to improve performance by integrating specialized packet processing hardware, and reducing costs by making the specialized hardware programmable (Comer, 2004). However, most existing work on network processors concentrates on more traditional networking operations like routing (Kalin and Peterson, 2001; Spalink et al., 2001) and active networking (Kind et al., 2003), while only a few approaches have been proposed in the area of multimedia applications. One example is the booster boxes from IBM (Bauer et al., 2002) which try to give network support to massive multiplayer on-line games by combining high-level game specific logic and low-level network awareness in a single network-based computation platform. Another example is a video quality adjustment mechanism (Yamada et al., 2002) that is implemented on a network processor, reducing the transmitted video quality for less capable links and client receivers.

3 IXP1200 Overview

The Intel IXP network processors (Intel Corporation, 2001; Comer, 2004) are examples of fourth generation network processors. Figure 1 shows the IXP1200 network processor that for example is integrated on the enp2505 card (Radisys Corporation, 2002) that we have used in our system. The basic features of this network processor include four 100 Mbps Ethernet interfaces, a general purpose 232 MHz StrongARM processor, six 232 MHz special-purpose processors called microengines for packet processing, and three main types of memory that should be used for different operations according to access time and bus bandwidth, i.e., 256 MB SDRAM for packet store, 8 MB SRAM for tables and stack manipulation, and 8 MB scratch (on-chip, not shown in figure 1) for synchronization and inter-process communication. The StrongARM is running a conventional Linux operating system and can be used as a traditional general purpose CPU. The microengines are compact RISC processors each running four concurrent threads where each has an own set of registers to make switches efficient.

4 Design

Multimedia applications are usually characterized by high bandwidth requirements and/or timing constraints. In our previous and current work on media-on-demand and interactive game systems, we have seen that the workload on intermediate nodes might be very resource consuming and that the processing results in increased latencies. On the other hand, Mackenzie et al. (2003) tested and analyzed the enp2505 and found many unused resources on the card. Similar

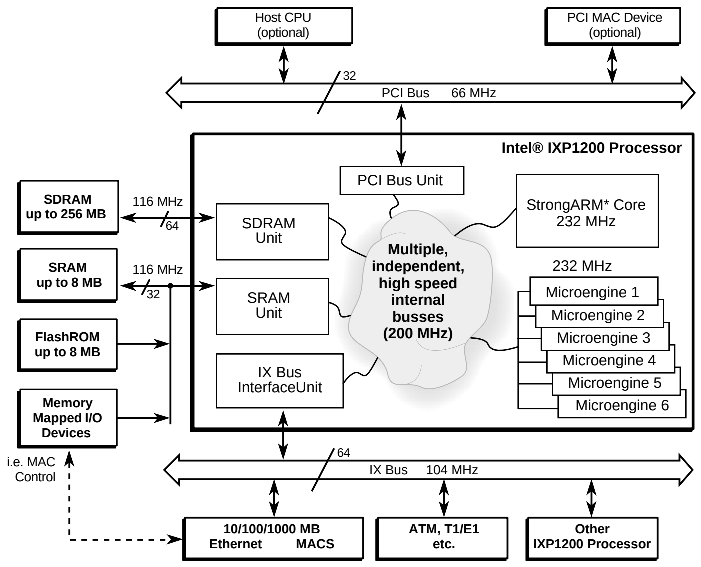

Figure 1: **Simplified block diagram of the IXP1200 (Intel Corporation, 2001)**

results have been found by Spalink et al. (2000) using the IXP1200 evaluation board (having eight 100 Mbps Ethernet interfaces). Their results show that forwarding of IP packets can be done using only 65% of the available capacity, and the performance could be further improved by increased memory bandwidth. Thus, these experiments indicate that movement of other operations like application specific processing to the network processor should be possible.

4.1 Multimedia Proxy Caches

In a traditional architecture (see figure 2A), all packets arriving at a proxy cache are processed on the host machine. This results in considerable resource consumption and latency due to bus transfers, interrupts, memory copying, checksumming and possibly application level processing operations. To increase the efficiency of intermediate nodes, such as proxy cache servers in our multimedia scenario, we design a system based on the observations above, i.e., programmable network processor cards are capable of low-latency forwarding and some application-specific processing while reducing the resource consumption of the host machine. Our proposed component (see figure 2B) therefore makes use of the existing on-board memory, the low-level packet processing microengines and the conventional StrongARM CPU to establish a faster and less resource-consuming data path in our proxy cache design.

Some of the basic features of our proxy include support for packet forwarding, data caching and overlay multicast. Each of these are briefly described in the next subsections, before we present the implementation and the results of our initial prototype.

4.2 Data Forwarding

If a data packet is passed through an unmodified proxy and forwarded directly to clients without any caching or data manipulating operations, the packet is still sent from the network card to the host and back to the network card (see thin arrows in figure 2A). Such an operation is resource demanding and introduces considerable latencies. This can be devastating in applications with low delay requirements like interactive on-line games or other virtual environments.

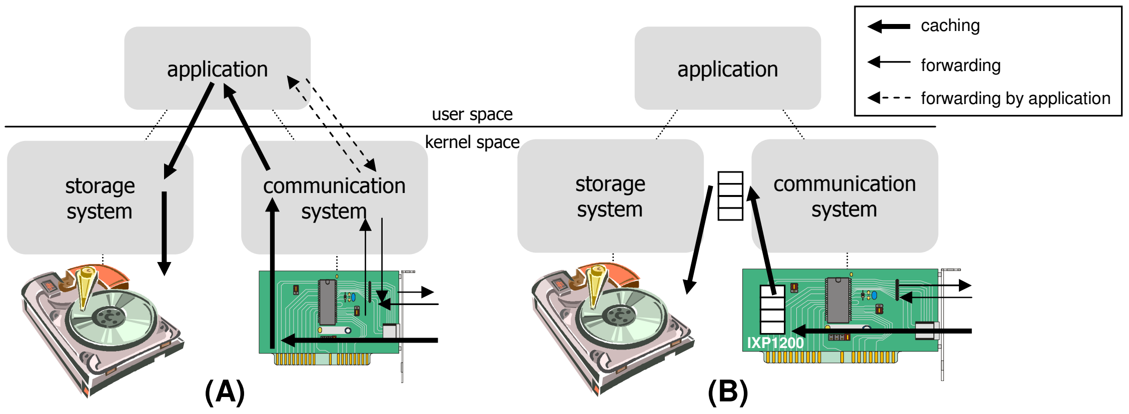

Figure 2: **Traditional (A) and network processor based (B) data paths**

Using a network processor, this operation can be performed much more efficiently and offload the host CPU at the same time. The network processor card's microengines are specialized, low-level units for packet processing and can efficiently identify and classify the packets. To exploit this, we use a specifically-created fast path forwarding component running on the microengines that determines whether a packet can be forwarded directly and if possible, sends it directly to the outgoing network interface (see thin arrows in figure 2B).

4.3 Caching

Obviously, caching a data stream to serve future requests with less access latency, server load and backbone load must involve the host machine. In an unmodified machine, each received packet is transfered to the host individually in a data transfer from network card to disk including several copy operations (see thick arrows in figure 2A) in addition to the processing overhead mentioned above. These data copy and processing operations are expensive in terms of bus, disk and memory bandwidth.

If the data can be stored directly on disk (without modification by an application), a first optimization can be to make a fast in-kernel data path on the host. However, an optimal zero-copy approach can not be used for two reasons. One is that packets are smaller than blocks that can be written to disk efficiently, and that they must therefore be assembled into a larger block before the disk write operation, which means that this approach generates one interrupt per arriving packet and one additional copy operation. The other is that packets can be lost or arrive out-of-order, which leads to fragmented writing to disk unless the packets are reordered in memory first.

Our proposed component (see thick arrows in figure 2B) aims at offloading the packet processing and supporting an optimized data movement operation by using the on-board hardware. Packet processing is performed on-board, and to reduce the number of interrupts and to collect data to be stored as larger items on disk, we queue several packets on the card. When sufficient data is received to make an efficient disk operation, we use one interrupt to inform the host that data has arrived. As packet processing has already been performed we can also use scatter-gather DMA to transfer all the fragmented memory objects in one bus operation into a larger contiguous memory area, and finally, these data can be sent directly, without further copy operations, to the storage system for large efficient disk write operations.

4.4 Overlay Multicast

In several applications, there might be several receivers of a data element, e.g., in interactive on-line games where all players want the same updated status information. However, due to the lack of IP multicast support in most of the Internet today, there exist many approaches to application level / overlay multicast. To minimize overhead and latency, we wish to perform this operation on the network processor by having a component that can send a data element to several receivers.

4.5 Integration

Integrating all this functionality into one system may raise several challenges, but one of the most important in order to make the operations on the card efficient is to support efficient sharing of data where packets are to be delivered to several destinations concurrently, e.g., both forwarding to several clients and caching to disk. These operations can be performed in several ways. A first step is to do this sequentially by reusing the same data element, but this introduces large delays for the destinations that are last in the sequence. An alternative is to make copies of the packet and to send each copy to a separate destination, but this results in overhead for the copy operation, which is also expensive. Instead, we wish to combine these, use one copy of the data and send to several destinations in parallel. The problem of having a sub-component releasing the memory holding the packet is solved by using a reference counter, and releasing memory only when the packet has been forwarded to all destinations.

5 Initial Implementation

As a proof of concept, we have implemented RTSP signaling and RTP forwarding as depicted in figure 3. We have used the ingress and egress active computing elements (ACEs) in the SDK from Intel to receive and send packets on the microengines, respectively. After arriving at the ingressACE, the packets are classified on the microengines using our classification and forwarding ACE. If the packet is to be forwarded, it is simply passed on to the egressACE with the necessary header modifications and from there, sent out onto the network. If a packet is classified as an RTSP packet, an RTP packet that should be cached (only interface implemented) or other packets requiring more processing, the packet is sent to the StrongARM. On the StrongARM, the packet is processed using another existing ACE implementing the protocol stack (stackACE), and finally, if it is an RTSP packet, it is processed in the RTSP server process running in the Linux environment.

To test the RTSP control functionality and RTP forwarding, we ran a Darwin streaming server from Apple and used a simple client to request and retrieve a small QuickTime movie from the server. The server sent the movie using 1024 bytes RTP packets, and in between, we set our proxy prototype up to see how the offloading performed.

6 Results and Evaluation

Our experiments, using the Darwin server to stream a QuickTime movie and running the RTSP proxy server and the packet classifier/RTP forwarder on the IXP, show that the prototype can successfully offload both control and data plane of a proxy cache to the network processor in a forwarding scenario. The RTSP server runs on the on-board StrongARM which should give faster responses to requests in addition to offloading the host machine. Additionally and more

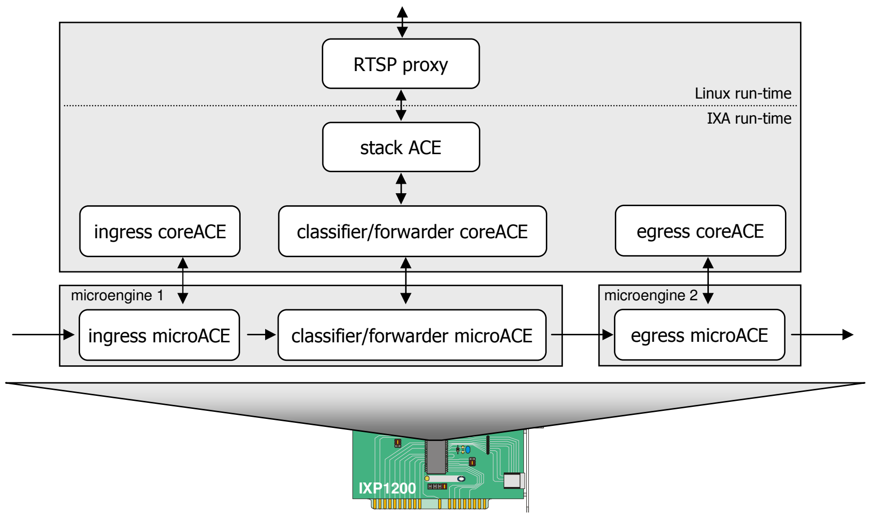

Figure 3: **ACE layout**

importantly, as the number of data packets outnumbers the number of control packets by an order of magnitude, it also showed that it is possible to build a fast data path using microengines only. In the forwarding scenario, we offloaded all expensive operations from the host machine, and it greatly reduced the total time (intermediate node latency) to forward a packet.

To measure the number of cycles to process and forward an RTP packet, we used the on-board cycle count register. We inserted cycle counter instructions at several places, i.e., looking at figure 3 we inserted the probes before the ingressACE, between the ingressACE and our classifier/forwarder ACE, after the classifier/forwarder ACE and after enqueuing the packet for the egressACE[1]. Our processing overhead and forwarding latency results are given in table 1 for each component, and a plot of the experienced cycle consumption of the approximately 3200 first packets of the stream is depicted in figure 4.

Prototype component	*cycles*	*microseconds*
Ingress processing	493	2.13
Classification, forwarding and header modification	644	2.78
Enqueuing for egress	194	0.83

Table 1: **Average overhead in the prototype**

As seen in the plot, most packets are processed in about 1325 cycles, but to explain the variation, we performed some more experiments. In these tests, we tested the forwarding operation as an atomic operation, i.e., no other thread ran on the processor, by blocking on every memory access on the non-preemptive microengine scheduler, and we experimented with the different memory queues as this is a resource shared by all microengines. Our results show that only the maximum r in the original measurement was higher than the atomic execution, and the variation is slightly reduced when using prioritized memory accesses. Our best explanation for the variance is therefore a difference in memory latencies and scheduling of the other threads running on the same microengine. However, there still is some variance that cannot be explained by our test results, and more experiments should be performed.

[1]The processing performed by the egress component is not measured, because this is running on a separate microengine. However, more detailed and comprehensive experiments are scheduled.

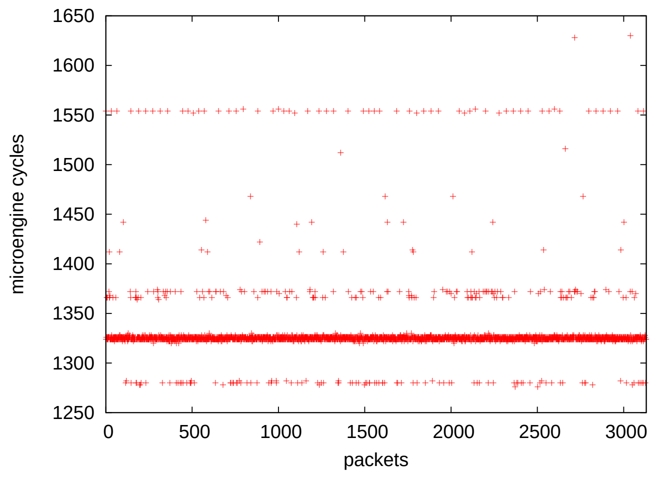

Figure 4: **Experienced packet cycle count consumption**

To see the real gain compared to traditional packet processing, we have compared our experiments with the processing and forwarding experiments on a 2.0.34 Linux kernel described in (Guo and Zheng, 2000)[2]. As we have not measured the sending operation (egressACE), we look at the reception, classification and forwarding operation which used about 1150 cycles (or 5 microseconds) on our network processor. Similar operations in the Linux stack, i.e., kernel communication system processing only, without data movement to the application and application specific operations, consume 12250 cycles. Additionally, more recent results also show a considerable challenge in dealing with forwarding latency on existing systems today. In (Liu et al., 2005), the average processing delay of overlay nodes in a P2P application was found to be approximately 30 milliseconds. Compared with the two to ten milliseconds physical network latency between the hosts in that test, this contributes significantly to the overall end-to-end delay. Thus, we have a considerable reduction in both processing latency and resource usage compared to systems involving the host in the packet forwarding operation.

Finally, as in (Spalink et al., 2000; Mackenzie et al., 2003), our tests show that we should have resources available to perform even more processing on-board. The four 100 Mbps interfaces on the enp2505 can sustain a maximum data-rate of 400 Mbps. Furthermore, the maximum throughput of our forwarder using 7.9 microseconds per 1024 byte packet (assuming equal ingress and egress processing overhead) is 1036 Mbps, i.e., resources for other processing are available as we have so far neither employed parallelism, nor utilized more than 2 microengines – one for ingress, classification and forwarding and one for egress.

7 Conclusion and Future Work

In this paper, we have investigated a design for a multimedia proxy cache offloader using the Intel IXP1200 network processor. Our prototype successfully offloads the host CPU with RTSP control functionality and RTP forwarding, i.e., performs operations in both the control and data plane. Our results show that a lot of resources can be freed on the host and at the same time make the system as a whole more efficient by enabling a faster, and less resource-demanding data path. Specifically, we found that a data item can be forwarded in a fraction of the time

[2]We have an ongoing study of the performance profile of the network stack and the forwarding operation in Linux and NetBSD, but it is not finished.

compared to traditional systems moving all data through the host CPU.

With respect to ongoing work, we are currently working on adding support for the designed caching operation[3] and are going to perform more extensive tests and evaluations. With respect to future work, we are looking into the possibility to test the system on much faster versions of the IXP processors, and there are many possible and interesting areas in which to extend the current prototype. In addition to adding the caching and multicast services, this includes efficient communication between the NIC and the host over the PCI bus and looking at the design in other distribution architectures, such as cooperating proxies, multi-level proxies and P2P architectures.

References

D. Bauer, S. Rooney, and P. Scotton. Network infrastructure for massively distributed games. In *Proceedings of the Workshop on Network and System Support for Games (NetGames)*, pages 36–43, Braunschweig, Germany, Apr. 2002.

D. E. Comer. *Network Systems Design using Network Processors - Intel IXP version*. Prentice-Hall, 2004.

C. Guo and S. Zheng. Analysis and evaluation of the tcp/ip protocol stack of linux. In *Proceedings of the IEEE International Conference on Communication Technology (WCC-ICCT)*, pages 444–453, Beijing, China, Aug. 2000.

S. Kalin and L. Peterson. Vera: An extensible router architechture. In *Proceedings of the IEEE International Conference on Open Architectures and Network Programming (OPENARCH)*, pages 3–14, Anchorage, AK, USA, Apr. 2001.

A. Kind, R. Pletka, and M. Waldvogel. The role of network processors in active networks. In *Proceedings of the IFIP International Workshop on Active Networks (IWAN)*, pages 18–29, Kyoto, Japan, Dec. 2003.

L. S. Liu, R. Hampole, B. Seo, and R. Zimmermann. Active: A low latency p2p live streaming architecture. In *Proceedings of SPIE/ACM Conference on Multimedia Computing and Networking (MMCN)*, San Jose, CA, USA, Jan. 2005.

K. Mackenzie, W. Shi, A. McDonald, and I. Ganev. An intel ixp1200-based network interface. In *Proceedings of the Workshop on Novel Uses of System Area Networks (SAN-2)*, Anaheim, CA, USA, Feb. 2003.

Intel Corporation. Intel ixp1200 network processor family – hardware reference manual, Aug. 2001. URL `http://www.intel.com/design/network/manuals/278303.htm`.

Radisys Corporation. ENP-2505/2506 data sheet, Jan. 2002. URL `http://www.radisys.com/files/-1160_04_1202.2.pdf`.

T. Spalink, S. Kalin, and L. Peterson. Evaluating network processors in ip forwarding. Technical Report TR-626-00, Computer Science Department, Princeton University, NJ, USA, Nov. 2000. URL `ftp://ftp.cs.princeton.edu/techreports/2000/626.pdf`.

T. Spalink, S. Kalin, L. Peterson, and Y. Gottlieb. Building a robust software-based router using network processors. In *Proceedings of the ACM Symposium of Operating Systems Principles (SOSP)*, pages 216–229, Banff, Alberta, Canada, Oct. 2001.

T. Yamada, N. Wakamiya, M. Murata, and H. Miyahara. Implementation and evaluation of video-quality adjustment for heterogeneous video multicast. In *Proceedings of the Asia-Pacific Conference on Communications*, pages 454–457, Bandung, Indonesia, Sept. 2002.

[3]We also have current activities using the network processor in similar projects, like a server cluster and packet priority management.