An Approach Towards Enabling Intelligent Networking Services for Distributed Multimedia Applications

Srikanth Sundaragopalan, Ada Gavrilovska, Sanjay Kumar, Karsten Schwan

{srikanth, ada, ksanjay, schwan}@cc.gatech.edu

Center for Experimental Research in Computer Science
Georgia Institute of Technology, Atlanta, Georgia 30332-0280

Abstract

An increase in network speeds and addition of new services in the Internet has increased the demand for intelligence and flexibility in network systems. This paper explores the extent to which an emergent class of programmable networking devices – network processors, can be used to deliver more efficient, innovative services to multimedia applications. We present and experimentally evaluate an architecture model, which enables the dynamic creation of application-specific services on programmable communication devices, using the Intel IXP2400 network processor and an image manipulation application.

Keywords: network processors, IXP, IP Multicast, streaming applications

1. Introduction

An increase in speed of networks and addition of new services in the Internet has increased the demand for intelligence and flexibility in network systems. Distributed multiplayer video games based on distributed virtual environments are becoming increasingly popular. These applications involve periodic exchange of messages between the participating machines using some distributed agreement protocol. On the other hand distributed streaming applications involve large volumes of audio/video data to be transmitted across a wide range of clients [6]. Both these class of applications need two kinds of strict requirements: (1) bandwidth to send large amount of data, and (2) processing power in participating hosts to ensure timely processing and distribution of the content on continuous basis. These applications typically rely on the existence of an underlying communication infrastructure, such as application-level overlays or peer-to-peer networks. Application-specific processing actions are executed at intermediate nodes to implement data distribution functionality and ensure end-user QoS requirements.

Recent advancement in the network speed (Gigabit networks) has far reduced our concern about the bandwidth needed for such applications. However, the increase in sustainable bandwidth continues to present challenges. First, the growth in the network speed is fast approaching the memory interface speeds of general-purpose processors, making even a single memory reference impact the performance of the network application. Though the network is able to deliver data at high speeds, the overlay host machine becomes a bottleneck in processing and delivering them. Second, the increase in diverse mobile and wireless end-user platforms has created demand for continuous delivery of application services in resource poor environments (e.g., low-bandwidth connectivity, and limited computation power). In order to meet the high-bandwidth, low and predictable latency requirements of multimedia applications, additional processing needs to be applied to the data path in the overlays.

We argue that such processing and distribution of high bandwidth data must be carried out in the *fastpath* to get the maximum performance. The fastpath processing could be entirely supported by dedicated hardware, if only for their cost and inflexibility. With emergence of high-speed programmable network processors (NPs), software based solutions are considered feasible compared to their costly hardware based counterparts. Performance of previous implementations of software based router and QoS has strengthened the cause of exploring various services that can be added at network processor level [1,4,5,8]. Network processors such as IXP 2xxx series are one such high-speed programmable processor, which meets these requirements with less cost and greater flexibility through their programmable nature.

This paper presents and evaluates a flexible architecture which supports efficient execution of

variety of services required in distributed media applications.

2. Distributed Media Services

A wide range of multimedia services could be supported at the network level with the use of high-speed network processors. We can broadly classify these services as 1) *path transforming services (PathX)* -- manipulation on the media stream, and 2) *data transforming services (DataX)* -- manipulation of the data content.

The first set includes services such as routing, multicasting, broadcasting, prioritizing, filtering, etc. For example, removing certain frames from an MPEG stream to produce an MPEG video with reduced, but still satisfactory quality can be easily supported on these platforms. All of these services would require access and manipulation of the protocol and application level headers, but still leave the data content unmodified.

The second category includes services such as transforming/transcoding the data in different format representation, down sampling image quality by modifying the image, cropping the image to match to client view, compression algorithms, encryption, etc. Applications which require such functionality at the network level, include visualization of remote experiments, or camera-captured data for a pool of clients with wide range of interests in the ongoing visualization, and/or which use devices with varying computational or networking capabilities. These applications need services that (1) render appropriate view of imaging data on behalf of the clients, and (2) match the quality of the image stream to the client's interests, or device and connection capabilities.

Finally, consider services like application-level IP multicasting, needed in variety of distributed media applications. In these services, once the data is received, it needs to be duplicated to 'n' other members, as indicated in the routing tables. Such duplication involves heavy copying, and imposes additional loads on the host's memory and I/O infrastructure. Moreover these packets need to be received by the network adapter, processed through the IP and UDP/TCP stacks and then delivered to the application, which manages multicast routing functionality. One could remove this multi layer approach by handling such duplication at the network level itself. One approach could be to program such multicasting logic into programmable networking devices such as NPs. Such programmable processors provide sufficient headroom for carrying out application level packet manipulation and in the fast path [4,2].

2.1. IXP 2xxx Architecture

The proposed architecture assumes the availability of programmable communication cores or network cards at (at least some) nodes participating in the distributed infrastructure. Our implementation uses network processors from the Intel IXP family, specifically the IXP2400, attached to standard Linux-based hosts, to represent these future platforms. We assume that any network processing platform will have similar architecture and functionality of Intel IXP 2400. The feature that is important from the perspective of our design is the presence of multiple processing engines (such as the microengines of IXP2400), which can be pipelined in arbitrary manner to perform certain tasks. We briefly outline the key architectural features of the IXP2400 architecture. For more detailed information on the IXP2400 refer to [3].

The Intel IXP2xxx network processors have specialized hardware to support network operations, which gives the ability to implement network services with high packet throughput and low latency. Apart from network centric operations, it also provides support for data processing in the form of CRC checksum calculations, integer arithmetic etc. The IXP2400 chip includes eight 8-way multithreaded microengines for data movement and processing, an Xscale core for management and other functions, local SRAM and DRAM controllers and an integral PCI interface with three DMA channels. The Radisys ENP2611 board on which the IXP2400 resides includes a 600MHz IXP2400, 256MB DRAM, 8MB SRAM, a POS-PHY Level 3 FPGA which connects to 3 Gigabit interfaces and a PCI interface. An Xscale core, running Linux, is primarily used for initialization, management and debugging. The IXP2400 is attached to hosts running standard Linux kernels over a PCI interface. Data is delivered to and from the host-resident application components through the IXP's network interfaces.

Apart from the above-mentioned features, there is wide range of tools (simulator, debuggers, etc.) available for application developers.

3. System Architecture

The high-level view of the distributed infrastructure targeted by our work can be described as follows. Media data is exchanged on continuous basis on top of an application-level overlay. The data may traverse intermediate nodes on the path from sources to destinations. The processing at these intermediate nodes may involve solely overlay routing actions, or may require additional application-specific data content and path manipulations. Based on the

processing actions required, or the existing loads at intermediate nodes, the data path may be mapped to their dedicated communication subsystems, at the network interface level. The mapping of processing actions to individual nodes in the overlay is responsibility of the middleware utilized by the distributed applications, and is not the focus of this paper.

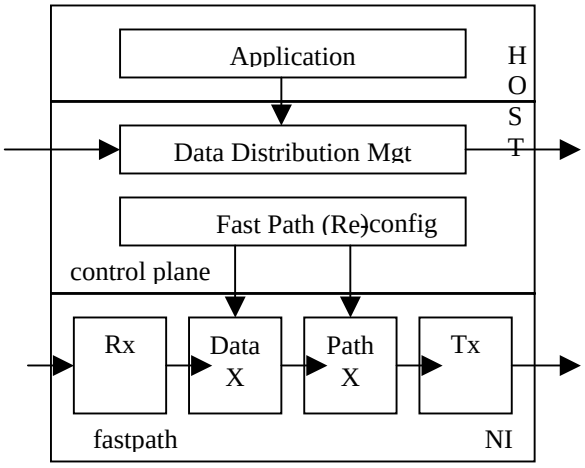

Fig. 1: System Architecture Overview

Individual nodes enhanced with programmable communication interfaces, similar to those discussed in [2] are depicted in Figure 1. In addition to data distribution functionality, the general-purpose host CPU may execute other application components. The distribution of data in the overlay, and deployment of processing on behalf of overlay clients is managed by a Distribution Manager. This layer drives any reconfiguration actions through which the data path through this node can be offloaded onto the communication interface (e.g., programmable NIC, attached NP, dedicated communication core, etc.).

The core functionality implemented on the fast path is encapsulated by the Rx and Tx tasks, which implement receive- and transmit-side protocol processing. Additional DataX and PathX tasks may be deployed on the fast path to implemented new services, thereby forming a processing pipeline. Depending on the underlying platform architecture and resource availability, DataX and PathX tasks are mapped to separate execution contexts (e.g., threads, microengines, processing elements). This may be particularly needed for DataX tasks, which have greater computational requirements and include repeated access to data content stored in chip memory. PathX tasks may be combined and executed jointly with the Tx processing. Compositions of DataX and PathX tasks can be formed to represent variety of services, ranging from data filter, transcoding, or multicast [2,6].

The processing executed as part of the DataX and PathX blocks can be dynamically configured through the host-resident FastPath Configuration interface.

4. Implementation Detail

As an instance of the pipelined architecture, we have implemented a Packet Forwarding Application and an Image Filtering Application to benchmark application level data manipulation in fastpath. Our current implementation assigns individual blocks from the fastpath pipeline to separate microengines on the IXP2400 NPs. Communication between pipeline stages is implemented through controlled access to shared ring buffers. SRAM based data descriptors describe the application level data, stored across multiple DRAM resident buffers. Communication with the host-resident control processes occurs via using shared mailboxes implemented on top of a PCI-based interface, similar to [2]. Fastpath reconfiguration requires involvement of the Xscale core, and has already been implemented for the previous generation IXP NPs with practically negligible service interruption (28-30 microseconds). Additional implementation details appear in the extended version of this paper [7].

5. Performance Analysis

We have carried out different sets of experiments to analyze the performance of application level data manipulation.

Experimental setup. We perform the experiments in two different setups. The first is purely host-based and the data path from source to destination traverses an intermediate general-purpose host. In the second setup, the data path traverses an IXP NP on its route to destination. In each case, additional processing is applied to the data at the intermediate node. Our host machines are 4 CPU with Intel Xeon processors, each with 2.4 GHz clock speed. The IXP test-bed uses the aforementioned Radisys ENP2611 boards, with eight microengines running at 600 MHz each. In order to simulate additional processing on the intermediate nodes in a distributed infrastructure, we run a CPU intensive process on the host machines, the `applu` application from the CPU2000 benchmark suite. The experiments use data streams generated from a flow of PPM images of varying sizes (which span multiple Ethernet frames), transmitted at maximum rates.

In the host based experiments, the image streams are sent from the source host to the intermediate one via UDP. Here, the user level application component waits for the entire application-level data to arrive

(i.e., entire image) and then either forwards or grey scales the image and sends it to the destination host. At the destination, we either use raw sockets to capture Ethernet frames or receive the data using UDP depending on the performance measurement needed. For latency measurements we use UDP at the destination, but for measuring the throughput we need to capture Ethernet frames of these UDP packets, and therefore use raw sockets.

For the IXP based test-bed, the image is packetized into Ethernet frames and is pumped into the LAN through raw sockets. The IXP network processor receives them and either forwards or grey filters the image before dispatching it to the destination host. At the destination, we again use raw socket application to capture the Ethernet frames.

End-to-End latency. PPM images are fragmented into 1500 bytes Ethernet frames and are put into the LAN using the raw sockets. The first timestamp is recorded when all the Ethernet frames are sent into the LAN and the second one when we receive all the frames at destination. We repeat the same experiment with the source and destination interchanged. This is done in order to remove the discrepancies in the two host machines clocks. We average the results from these measurements to get the actual end-to-end latency between these hosts. The experiment is carried out both on the host based test bed and the IXP based test bed. Table 1 shows the end-to-end latency for images of different sizes. The results show that, there is a substantial latency difference between host-based test bed and the IXP-based test bed for images of lesser size and significantly less difference when the image size is big. This is because the kernel/user space overhead and the stack-processing (UDP/IP) overhead are quite significant compared to the processing carried out on the data content itself.

Image size (B)	Number of frames	IXP latency (usec)	Host latency (usec)	Latency decrease (%)
57616	41	7150	8290	16
14436	11	1935	2956	52
3636	3	621	1499	141
936	1	340	1256	269

Table 1. End-to-end latency for the IXP-based vs. Host-based test-bed for grey scaling service

Throughput. Here again, we evaluate a PathX forwarding task, and a DataX grey scaling task. At the destination host, we measure the total time to receive the entire image stream, and compute the attained throughput. The results are presented in Table 2. In both case the IXP-based implementation of the service outperforms the host-one. Simulation measurements indicate that the IXP-based implementation will significantly outperform the host-based implementation under increased network loads, in spite of the disparity of the computational capacity of the host CPUs vs. the IXP microengines.

	IXP-based (Mbps)	Host-based (Mbps)	Gains (%)
DatX	94.7	93.9	1
PathX	94.5	91.8	3

Table 2. Through measurements for the host- vs. IXP-based test-bed.

6. Conclusions

Technology advances have created the ability to bring intelligence into the networking infrastructure, and this work explores one architecture which, using off-the-shelf communication devices, can deliver improved levels of support for a range of services needed in distributed multimedia applications.

Acknowledgment: Ramkumar Gandhapuneni and Prashant Thakare worked on the original implementation of the imaging application.

7. References

[1] A. Gavrilovska, S. Kumar, K. Schwan, The Execution of Event-Action Rules on Programmable Network Processors, *OASIS 2004, held with ASPLOS-XI*, Oct. 2004.

[2] A. Gavrilovska, K. Schwan, O. Nordstrom, H. Seifu, Network Processors as Building Blocks in Overlay Networks, *Hot Interconnects*, 2003.

[3] IXP Intel Network Processor Family, http://developer.intel.com/design/npfamily

[4] T. Spalink, S. Karlin, L. Peterson, Y. Gottlieb, Building a Robust Software-Based Router Using Network Processors, *SOSP 2001*.

[5] Y-D. Lin, Y-N. Lin, S-C. Yang, Y-S. Lin, DiffServ over Network Processors: Implementation and Evaluation, *Proc. of Hot Interconnects 10*, Aug. 2002.

[6] S. Roy, J. Ankcorn, S. Wee, An Architecture for Componentized, Network-Based Media Services, *Proc. of IEEE International Conference on Multimedia and Expo*, Jul, 2003.

[7] S. Sundaragopalan, A. Gavrilovska, S. Kumar, K. Schwan, An Approach Towards Enabling Intelligent Networking Services for Distributed Multimedia Applications, CERCS Technical Report, GIT-CERCS-05-12, Apr. 2005

[8] X. Zhuang, W. Shi, I. Paul, K. Schwan, Efficient Implementation of the DWCS Algorithm on High-Speed Programmable Network Processors, *Proc. of Multimedia Networks and Systems (MMNS)*, Oct. 2002.