QoS-Sensitive Protocol Processing in Shared-Memory Multiprocessor Multimedia Servers (Extended Abstract)

Ashish Mehra Kang G. Shin

Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48109-2122
Email: {*ashish,kgshin*}@*eecs.umich.edu*

1 Introduction

The advent of high-speed networks and demand for distributed multimedia applications now requires that the communication subsystems in end hosts (or simply hosts) be designed and implemented to facilitate and provide certain quality-of-service (QoS) guarantees. An application specifies its QoS requirements for *each* active connection in terms of several parameters such as end-to-end delay, delay jitter, and bandwidth; additional requirements regarding packet loss and in-order delivery can also be specified [1,2]. Additionally, the application may need to specify its traffic characteristics on each connection in order for the host communication subsystem and network to provide the required guarantees. Acceptance of a guaranteed-QoS connection by the network implies a contract between the network and the application that the requirements of the latter will be met as long as the application honors its traffic specifications. Given appropriate support within the network to establish and maintain such connections, we focus on the problem of maintaining QoS-guarantees within the communication subsystem on the host[1]. In particular we consider shared-memory multiprocessor (SMMP) hosts since medium- to large-scale SMMPs are increasingly being employed as multimedia servers and desktop workstations.

Large-scale multimedia servers may support hundreds of guaranteed-QoS connections simultaneously, each requiring the processing and transfer of large amounts of data. Supporting multiple guaranteed-QoS connections on SMMP hosts requires a careful study of the hardware and software structure of the communication subsystem, which includes protocol processing and access to/from the network via the network adapter. The communication subsystem structure must facilitate a range of QoS guarantees, while keeping resource utilization high, because of three aspects of application behavior: (i) applications may have a variety of QoS requirements, (ii) there may be large variations in the traffic behavior of each connection even when traffic specifications are not violated, and (iii) the number of established connections and their desired QoS may vary over time. The primary goal, therefore, is to design the communication subsystem to make protocol processing and network transmission/reception *QoS-sensitive*.

In this paper we study the issues involved in designing the communication subsystem to provide QoS guarantees for connections originating or terminating at an SMMP host. In particular, we propose an architecture for QoS-sensitive protocol processing and highlight the design tradeoffs involved in realizing the architecture on an SMMP host. In the remaining sections we motivate the necessity of such an architecture and highlight its salient features and associated design tradeoffs. We conclude by describing our simulation-based evaluation framework which features object-oriented, parameterized models for resources participating in communication and QoS-sensitive resource management policies.

The work reported in this paper was supported in part by the National Science Foundation under grant MIP-9203895 and the Office of Naval Research under grant N00014-94-1-0229. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of NSF or ONR.

[1]End-to-end performance guarantees cannot be provided without maintaining QoS guarantees within the *computation subsystem* as well. While extremely relevant, the issues involved are beyond the scope of this paper. Since we focus on communication subsystem design for a server, we only consider transmission-side issues; our future work will consider reception-side issues as well.

2 Issues and Approaches for QoS-Sensitive Protocol Processing

Consider the problem of servicing a large number of guaranteed-QoS connections engaged in network transmission. In addition to such connections, there may be several best-effort connections that must be serviced in a fair manner, with reasonably good performance. The data to be transmitted over each connection resides either in an input device (such as a frame-grabber) or in host memory; we assume that the computation subsystem prepares outgoing data in a QoS-sensitive fashion before handing it over to the communication subsystem. Each guaranteed-QoS connection is assumed to have the following traffic flow semantics. On each connection data is transferred from the server to the destination client (unidirectional data flow), successive messages on a connection must be delivered in the order they were generated (in-order message delivery), and data that suffers loss of QoS guarantees within the network is unusable and hence not retransmitted (unreliable data transfer). Best-effort connections do not have the requirement of in-order delivery but may require retransmissions to ensure loss-less data transfer. These connection semantics are applicable to a large class of multimedia applications, and retain the generality of our proposed architecture for QoS-sensitive protocol processing.

2.1 Requirements

Once established, transmission on each connection must commence by a certain deadline and, once initiated, the connection must receive a certain rate of protocol processing and transmission bandwidth; both the deadline and the rate are derived from the connection's QoS requirements. While policies for uncontrolled sharing would maximize resource utilization, and hence be ideal for best-effort traffic, these sharing models are ill-suited for QoS-sensitive protocol processing. Uncontrolled sharing of communication resources introduces unpredictability in protocol processing and can result in violations of QoS guarantees. For example, connections may receive service in an order that causes a less urgent message to be transmitted ahead of an urgent one. Similarly, on a given connection, the order of message transmissions may violate the requirement of in-order message delivery. This is because, assuming static network routes for connections, in-order delivery of messages necessitates in-order message transmission. Therefore, the communication subsystem must exercise fine-grain control over management of resources involved in communication, such as processors and the network adapter.

Within the communication subsystem, the degree of sharing of processors amongst connections depends on the transport system architecture [3] and resource management policies employed. Resources such as processors and buffers must be allocated to individual connections in order to meet their QoS requirements. This allocation must attempt to reduce load imbalance, achieve high resource utilization, and maximize the number of connections that can be serviced. Unlike uniprocessor hosts, network adapter access by multiple processors must be coordinated to satisfy the QoS requirements of individual connections. The nature and overhead of this coordination depends on the design features and performance characteristics of the network adapter. The interconnection architecture (e.g., bus or crossbar) and shared-memory model (uniform or non-uniform access) within the host also influence this aspect significantly.

A final requirement is that connections must be insulated from one another so that an ill-behaved connection only degrades its own performance and not that of any other connection. A connection can violate its traffic specification in two ways: initiating bursts of message transmissions and transmitting messages larger than the maximum message size allowed on that connection. The policies for resource management must ensure that an ill-behaved connection does not consume resources at the expense of well-behaved connections. At the same time, the ill-behaved connection must continue to receive service insofar as possible, perhaps with a degraded QoS, since the traffic specification violations are likely to be of a temporary nature.

2.2 Proposed Architecture

In order to satisfy the above requirements, we propose a communication subsystem architecture which provides QoS-sensitive protocol processing by

- *dedicating* a set of processors for protocol processing,

- *mapping* active connections to these protocol processors,
- *scheduling* protocol processing on each protocol processor, and
- *coordinating* (i.e., scheduling) access to the network adapter by the active protocol processors (i.e., those that have packets to transmit).

Together, these features allow the communication subsystem to exercise fine-grain control over resource management in order to satisfy QoS guarantees, as discussed below. We assume that all activities other than managing resources involved in communication, performing protocol processing, and coordinating access to the network link, are executed on the computation subsystem.

Dedicating a set of processors for protocol processing implies a *static partitioning* of the available processors in the host; this eliminates sharing (and the resulting interference) between the computation and communication subsystems. Provision of multiple protocol processors facilitates scalable server design by increasing the processing capacity of the communication subsystem and allowing concurrent handling of different connections. Protocol processing is based on a vertical process architecture employing the process-per-connection model [3]. Each connection has associated with it a unique process (the connection "handler") and a first-in-first-out (FIFO) queue of messages waiting to be processed and transmitted; the connection handler processes the queued messages one at a time. Maintaining a FIFO queue of messages per-connection preserves the order in which messages are generated on each connection; early arrivals due to message bursts are absorbed in the message queue and do not affect processing on other connections. A message is dequeued for processing only when the previous message has been completely processed and the corresponding packets enqueued for transmission. A message violating the message-size limit can be dropped as soon as it is generated. Alternately, the message can be enqueued for protocol processing but is processed at a reduced priority. We lower handler priority in proportion to the amount by which the message exceeds the size limit; the priority is reduced incrementally to spread out the total degradation evenly across the maximum-message-size "blocks" in the original message.

Associating a unique process with each connection simplifies the mapping of connections to processors and the scheduling of the corresponding handlers on each processor. When a connection is established, it is mapped to exactly one of the protocol processors (in other words, a processor is allocated to it). This mapping is a function of the available processing capacity on each processor and the QoS requirements of the connection to be established. Mapping a connection to exactly one processor ensures that packets belonging to a message are enqueued for transmission in the correct order; it also reduces coordination and synchronization overheads. Since more than one processor may have sufficient processing capacity to accommodate the new connection, processor selection is done based on heuristics such as first-fit, least-current-load, least-number-of-connections, or a combination thereof. This mapping could be changed each time a new connection is accepted for service. However, for connections reserved in advance (i.e., the client announces its connection requirements well before the time it actually needs the established connection), an optimal static mapping may be derived and processors scheduled accordingly.

Given a mapping of connections to processors, per-connection protocol processing is scheduled on each processor according to the QoS requirements of the connections mapped onto that processor. Both dynamic-priority and fixed-priority real-time scheduling algorithms can be employed to manage each processor. The scheduling algorithm used determines the number and type of connections that can be mapped onto a processor without violating QoS guarantees. The processor-connection mapping may need to be changed in order to accommodate a new connection, improve utilization, or reduce load imbalance; the feasibility of connection remapping depends on the potential benefits and associated cost of remapping.

Since multiple connections receive service simultaneously, more than one processor would compete for access to the network adapter if each connection handler initiated transmission. Coordination amongst active processors is necessary to ensure that the network adapter transmits packets based on the relative QoS requirements of the connections. Instead of directly invoking transmission, each connection handler inserts its packets into a *shared* set of *link packet queues* that determine the order in which packets from all connections will be transmitted. Initiation of transmission can either be done by the network adapter, if it has the required capability and intelligence to access the packet queues, or transmission can be initiated by a special *link scheduler* executing on one of the processors. Note that scheduling of packets may be required even if the network adapter is cognizant of connections and their QoS requirements, as in ATM networks.

Best-effort connections are handled differently in the proposed architecture. They must be serviced in a fair manner so that reasonably good performance is delivered to them even in the the presence of guaranteed-QoS connections. Unlike guaranteed-QoS connections, best-effort connections are maintained as a shared pool of work available to *all* protocol processors. Processing of messages on best-effort connections is given higher priority than processing early messages on guaranteed-QoS connections, even if per-connection protocol processing is work-conserving. Instead of idling, a processor processes best-effort connections; in addition to providing processing capacity for best-effort connections, this keeps processor utilization high. Generation of messages on guaranteed-QoS connections immediately preempts processing on best-effort connections. Fair servicing amongst best-effort connections is provided through FIFO or round-robin scheduling policies. To provide a certain minimum processing capacity to best-effort connections, a number of protocol processors can be set aside. Fairness in packet transmissions is ensured by giving best-effort packets priority over early-arriving packets when scheduling packet transmissions.

2.3 Design Tradeoffs

The myriad design tradeoffs that arise within the framework of the proposed architecture can be classified as follows:

Processor-connection mapping and remapping: While an optimal static mapping may be derived for connections reserved sufficiently in advance, the absence of *a priori* knowledge of connection requests necessitates use of heuristics to guide the mapping. For a given set of protocol processors, there is a fundamental tradeoff between load imbalance amongst processors and the utilization of each processor. Since the available heuristics exploit this tradeoff differently, the choice of the mapping heuristic has a significant impact on the number and type (determined by the QoS requirement) of connections that can be established for service. The choice of mapping heuristic also determines the potential benefit and incurred cost of dynamically remapping connections to "garbage-collect" sufficient processing capacity for a new connection.

Per-processor connection scheduling and per-connection protocol processing: The utilization of each processor is determined by the scheduling algorithm employed and the mix of connections mapped onto the processor. This in turn determines spare processing capacity and hence the set of connections that will be mapped onto the processor in the future. The tradeoffs involved depend greatly on whether handler execution is fully preemptive (immediate preemption), semi-preemptive (bounded-delay preemption) or non-preemptive. Note that these three differ primarily in the *preemption granularity*, which can be viewed as the number of packets processed between preemption points. The preemption granularity determines the worst-case preemption delay, which must be balanced against preemption overheads such as context switches and cache misses. Additionally, per-connection protocol processing could be work-conserving or non-work-conserving; this impacts the handling of message bursts on a connection. Assuming that guarantees of other connections are not violated, work-conserving protocol processing would improve processor utilization. Finally, the scheduling of connection handlers is also affected by the mechanisms employed to police traffic. For example, a handler should be blocked if the connection does not have any available slots in the link packet queues; this can happen either when a handler is "working ahead," or when a message violates the maximum-message-size limit.

Global coordination (scheduling) for network adapter access: Since mutual exclusion is necessary to deposit packets in the link packet queues, the overhead of packet insertion could get excessive. Depending upon the preemption granularity for protocol processing, each handler can amortize this overhead by inserting a block of packets instead of inserting each packet as soon as it is prepared for transmission. The characteristics of the adapter and the interconnection architecture of the host play a significant role in realizing the coordination amongst processors. Important adapter characteristics include provision of DMA capability, amount of on-board packet memory and organization of packet queues, ability to maintain per-connection QoS guarantees, and bounded, predictable latency to access the communication medium. The interconnection architecture determines the data transfer path from the source (processor, main memory, or an input/output device) to the network adapter, and how the available data transfer bandwidth on this path is shared amongst the connections. A bus-based interconnection limits concurrent data transfers; furthermore, QoS-sensitive consumption of transfer bandwidth may not be achieved simply through priority-based arbitration amongst processors. A crossbar-based interconnection provides additional concurrency and, coupled with support in the network adapter, can support DMA-based data transfer on multiple connections simultaneously. If the protocol stack decouples data transfer from control, and the adapter supports DMA to/from any source in the host, bandwidth can be consumed in the order in which packets are transmitted.

Number of protocol processors: The partitioning of processors between the computation and communication subsystems depends on the expected computational and communication load per connection, respectively. With advance reservation of connections, the optimal partition can be determined off-line. Without *a priori* knowledge of connection requests, however, determining the partition is ad hoc. Given the capacity of each processor, a static choice of the *number* of processors may not scale with the number of connections. Depending upon the tradeoffs identified earlier, it may be possible to allow *controlled sharing* of processors between the computation and communication subsystem. Additional protocol processors could be deployed only when necessary, i.e., when new connections are established and cannot be accommodated on the available protocol processors.

3 Evaluating the Design Tradeoffs

As a first step towards demonstrating the feasibility of the proposed architecture and evaluating the design tradeoffs highlighted, we have implemented a QoS-sensitive communication subsystem on a single protocol processor [4] residing in a small-scale bus-based multiprocessor host with a NUMA configuration [5]. The implementation has been done in the context of *real-time channels*, a paradigm for real-time communication services in packet-switched networks [6]. The communication executive on the protocol processor is derived from x-kernel 3.1 [7]; our implementation of the protocol stack decouples data transfer and control to minimize data copying. The implementation features a *process-per-channel* model with fixed-priority and earliest-deadline-first semi-preemptive scheduling of channel handlers; each channel handler relinquishes the CPU to a waiting higher-priority handler at evenly-spaced preemption points. Protocol processing can be work-conserving or non-work-conserving, with best-effort channels given processing and transmission priority over "work ahead" real-time channels. Link access scheduling is performed by a non-work-conserving link scheduler thread that runs at the highest possible priority. Channel handlers violating their traffic specification are prevented from consuming processing and link capacity either by blocking their execution or lowering their priority relative to the well-behaved channels. We are currently evaluating the implementation to determine a variety of cost-performance tradeoffs, such as preemption granularity/overheads, and the effectiveness of the scheduling and traffic policing mechanisms.

Based on the insights gained, we are designing a powerful, parameterized, object-oriented SMMP simulator to study the proposed architecture and design tradeoffs identified in this paper. The design of the simulator is similar to the one developed in [8], but specifically geared towards QoS-sensitive protocol processing. In particular, it features parameterized models for resources (processors, memory, processor–memory or processor–processor interconnect, network adapter), resource management and scheduling policies (connection admission control, processor-connection mapping, per-processor scheduling, link access scheduling), and resource usage (connection requests, per-connection traffic generation and protocol processing). The overall structure of the simulator is illustrated in Figure 1.

As shown in Figure 1, the user configures the simulator for a given set of experiments via a specification script written in the simulator's specification language. The user's specification is parsed by a front-end module to determine the policies, parameters, and architectural environment to be used for the experiments. The specification of resources includes the number and type of available resources, their performance characteristics, and the architectural configuration of the SMMP; the simulator instantiates resources based on this specification. Resource management specification includes the policies and heuristics to use for resource management and scheduling, which is used to select and configure generic resource management modules in the simulator. The specification of workload generation is used to determine the QoS requirements and traffic characteristics of individual connections, and the probability distributions for generation of connection requests and per-connection traffic. Finally, the data collection specification is used to select the performance parameters and metrics for which the simulator must collect data and accumulate statistics. At the completion of the experiments, the simulator outputs performance results such as the utilization of each resource, the delivered QoS on each active connection, and the number of connections accepted/rejected.

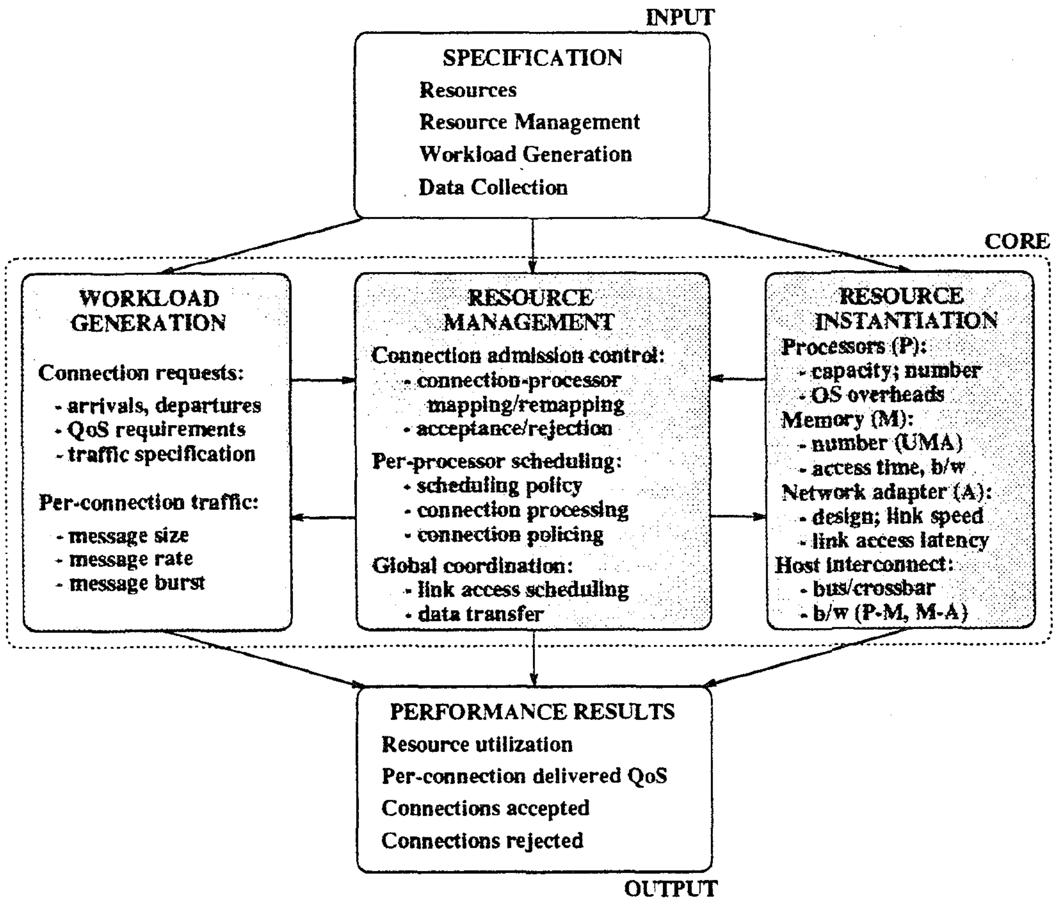

Figure 1: Structure of SMMP simulator

4 Related Work

The approaches taken to implement protocol processing in SMMPs essentially lie on two extremes. In one approach, each processor executing a process also performs protocol processing for messages transmitted by that process [9]. In this model, protocol processing is treated as work strictly local to each processor, resulting in an implicit sharing between the computation and communication subsystems. An alternative approach treats protocol processing as global work that can be scheduled uniformly on any available processor [10,11]; this results in explicit sharing between the two subsystems. These approaches may not suffice for QoS-sensitive protocol processing since they introduce unpredictability in the availability and allocation of processing resources, and complicate global coordination for network access. Our proposal for static partitioning of processing resources is similar to those for multiprocessor front-ends [12], except that a set of processors within the host are dedicated for protocol processing, as in [13].

Vertical process architectures employing process-per-connection and process-per-message models have been proposed to exploit parallelism in protocol implementations [3]. A process-per-message model seems unsuitable for QoS-sensitive protocol processing. Assuming that each message's shepherd process is independently scheduled on the protocol processors, simultaneous processing of multiple messages from the same connection would lead to out-of-order consumption of protocol processing and transmission bandwidth. Further, shepherd processes handling messages belonging to the same connection must now synchronize to maintain consistent connection state. With potentially several connections mapped to the same processor, it becomes more expensive to coordinate handling of messages on a connection and between connections. Our choice of the process-per-connection model and mapping of each connection handler to exactly one processor is based on these considerations. Recent results have shown that connectional parallelism delivers comparatively more scalable performance than message parallelism [14,15].

The design of high-speed network adapters, their performance characteristics, and implications for protocol stacks has received significant attention recently, for FDDI [16] as well as ATM [17,18] networks. However, the design tradeoffs have been explored primarily in the context of uniprocessor workstations and best-effort network traffic. We have studied the implications of network adapter characteristics for real-time communication on a Fibre Channel adapter manufactured by Ancor Communications [5].

References

[1] D. Ferrari, "Client requirements for real-time communication services," *IEEE Communications Magazine*, pp. 65-72, November 1990.

[2] M. Zitterbart, B. Stiller, and A. N. Tantawy, "A model for flexible high-performance communication systems," *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 4, pp. 507-518, May 1993.

[3] D. C. Schmidt and T. Suda, "Transport system architecture services for high-performance communications systems," *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 4, pp. 489-506, May 1993.

[4] A. Mehra, A. Indiresan, and K. G. Shin, "Design and evaluation of a QoS-sensitive communication subsystem," In preparation, July 1995.

[5] A. Indiresan, A. Mehra, and K. Shin, "Design tradeoffs in implementing real-time channels on bus-based multiprocessor hosts," Technical Report CSE-TR-238-95, University of Michigan, April 1995.

[6] D. D. Kandlur, K. G. Shin, and D. Ferrari, "Real-time communication in multi-hop networks," *IEEE Trans. on Parallel and Distributed Systems*, vol. 5, no. 10, pp. 1044-1056, October 1994.

[7] N. C. Hutchinson and L. L. Peterson, "The *x*-Kernel: An architecture for implementing network protocols," *IEEE Trans. Software Engineering*, vol. 17, no. 1, pp. 1-13, January 1991.

[8] K. Maly et al., "Parallel TCP/IP for multiprocessor workstations," in *Proc. IFIP TC6/WG6.4 Fourth Int'l Conf. on High Performance Networking*, pp. 103-118, December 1992.

[9] C. A. Thekkath, D. L. Eager, E. D. Lazowska, and H. M. Levy, "A performance analysis of network I/O in shared-memory multiprocessors," Computer Science and Engineering Technical Report 92-04-04, University of Washington, April 1992.

[10] A. Garg, "Parallel STREAMS: A multiprocessor implementation," in *Winter 1990 USENIX Conference*, pp. 163-176, January 1990.

[11] S. Khanna, M. Sebree, and J. Zolnowsky, "Realtime scheduling in SunOS 5.0," in *Winter USENIX Conference*, pp. 375-390, January 1992.

[12] A. N. Netravali, W. D. Roome, and K. Sabnani, "Design and implementation of a high-speed transport protocol," *IEEE Trans. Communications*, vol. 38, no. 11, pp. 2010-2024, November 1990.

[13] M. Bjorkman and P. Gunningberg, "Locking effects in multiprocessor implementations of protocols," in *Proc. of ACM SIGCOMM*, pp. 74-83, September 1993.

[14] E. M. Nahum, D. J. Yates, J. F. Kurose, and D. Towsley, "Performance issues in parallelized network protocols," in *Proc. USENIX Symp. on Operating Systems Design and Implementation*, pp. 125-137, November 1994.

[15] D. C. Schmidt and T. Suda, "Measuring the performance of parallel message-based process architectures," in *IEEE INFOCOM*, pp. 624-633, 1995.

[16] K. K. Ramakrishnan, "Performance considerations in designing network interfaces," *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 2, pp. 203-219, February 1993.

[17] B. Davie, "The architecture and implementation of a high-speed host interface," *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 2, pp. 228-239, February 1993.

[18] C. B. S. Traw and J. M. Smith, "Hardware/software organization of a high-performance ATM host interface," *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 2, pp. 240-253, February 1993.