A Network Co-processor-Based Approach to Scalable Media Streaming in Servers *

Raj Krishnamurthy, Karsten Schwan, Richard West
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
{rk, schwan, west}@cc.gatech.edu

Marcel-Cătălin Roşu
IBM T.J. Watson Research Center
mrosu@watson.ibm.com

August 14, 2000

Abstract

This paper presents the embedded construction and experimental results for a media scheduler on i960 RD equipped I2O Network Interfaces (NI) used for streaming. We utilize the Distributed Virtual Communication Machine (DVCM) infrastructure developed by us which allows run-time extensions to provide scheduling for streams that may require it. The scheduling overhead of such a scheduler is $\approx$ 65μs with the ability to stream MPEG video to remote clients at requested rates. Moreover, placement of scheduler action 'close' to the network on the Network Interface (NI) allows tighter coupling of computation and communication, eliminating traffic from the host bus & memory subsystem, allowing increased host CPU utilization for other tasks without being affected by host-CPU loading. Further,

*This work is supported in part by the Department of Energy under the NGI program and the National Science Foundation under a grant from Divison of Advanced Networking Infrastructure and Research, by hardware donations from Intel Corporation and software donations from WindRiver Systems (VxWorks).

NI-based schedulers are immune to host-CPU loading unlike host-CPU based media schedulers that are easily affected by transient load conditions. This makes media scheduling a viable and strong candidate for offloading on NIs. Architectures to build scalable media scheduling servers are explored - by distributing media schedulers and media stream producers among NIs within a server and clustering a number of such servers using commodity hardware and software. **KEYWORDS: clusters, networks, multimedia, embedded, QoS, OS, SAN, real-time, streaming, scheduling**

1 Introduction

Background. The scalable delivery of media and web services to end users is a well-recognized problem. At the network level, researchers have designed multicast techniques[1], media caching or proxy servers, reservation-based communication services[11] and media transmission protocols. For server hardware, scalability is sought by using extensible SMP and cluster machines[2, 7]. Scalability for server software is attained by using dynamic load balancing across parallel/distributed server resources, by using admission control and online request scheduling[27, 35] to control resource usage, improve throughput, and guarantee service for resources like CPUs[20, 5, 15], network links [33, 32, 35], and disks[7]. In addition, developers employ application-level or end-to-end solutions[16, 23] that adapt server and/or client behavior in response to user needs and resource availability. For instance, for clients, media caching/buffering and runtime variation of delivered service quality[30, 24] are two of many techniques that attempt to deal with limitations in client resources and with fluctuations in the service offerings experienced by clients.

Scalable Cluster Services. Our group is developing software solutions that aim to improve scalability in media servers, where servers are assumed to be clusters of processor/storage nodes connected via high performance system area networks[31]. A cluster is comprised of a switch and of network interfaces (NIs) connected to the cluster nodes. Each NI has a high performance host CPU-NI interconnect(e.g., a PCI bus), direct connections to the switch, a programmable CoProcessor supporting protocol processing, and local memory with direct connections to disk devices and other peripherals. The NIs used in our research include ATM FORE[26], Myrinet[31], and I2O-compliant network interface boards[14, 18, 13].

This paper employs a server configured as 16 quad Pentium Pro nodes connected via I2O-based NIs, each of which has two 100Mbps Ethernet links, a PCI interface to the host CPU, and two SCSI interfaces directly attached to disk devices. Consequently, host-to-host communications are supported by I2O board-resident protocols (like TCP and UDP), and media streams may flow from server disks to clients via host CPUs or directly via the I2O boards (see Figure 1)[18, 13].

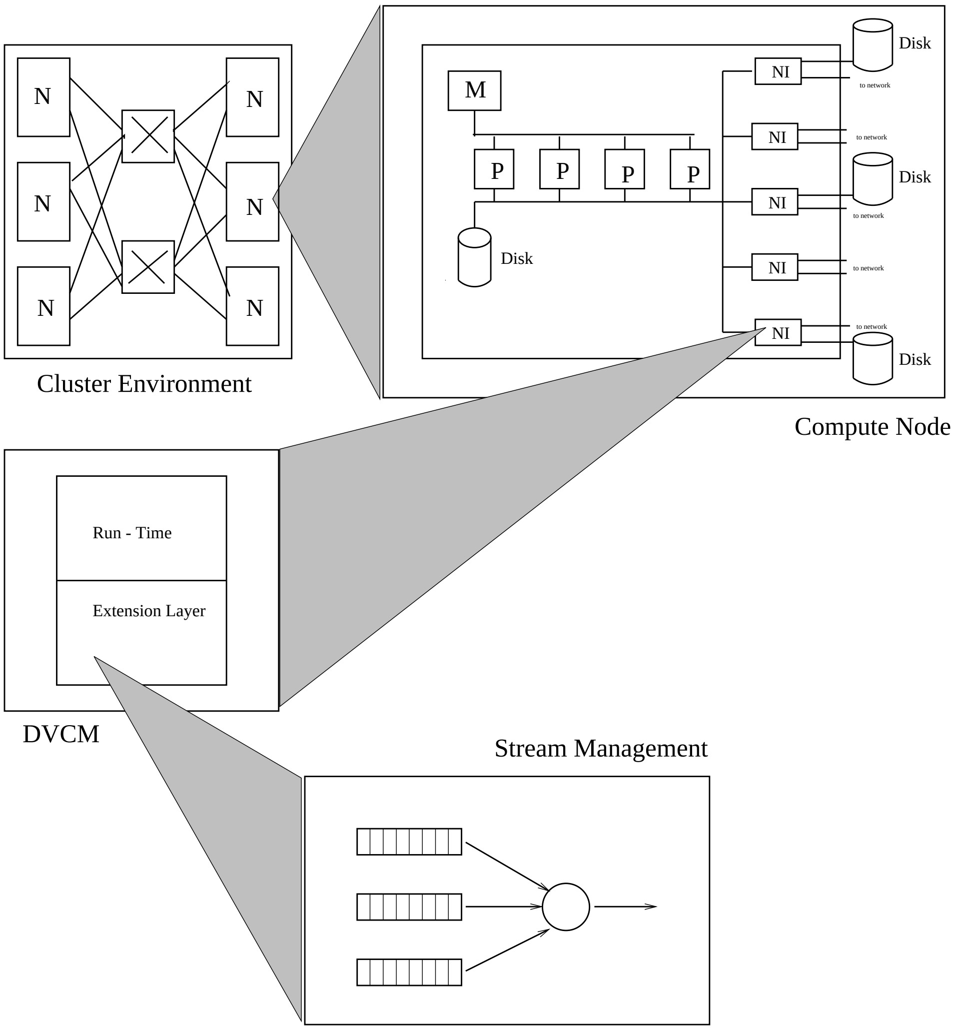

Figure 1: Cluster Hardware and Interconnect.

The approach to server organization we advocate is one that views a server like the one depicted in Figure 1 as an information processing, storage, and delivery engine that is programmed at two levels of abstraction, reflecting the hardware configuration being employed :

1. A cluster-wide, programmable *distributed virtual communication machine* (DVCM) executes 'close' to the network, on the CoProcessors as shown in Figure 1. The cluster-wide services executed by

this machine are available to nodes' application programs as *communication instructions*.

2. High-level node-specific services are implemented on host nodes. However, such host-resident application programs may also *extend* the DVCM with additional 'instructions' to support their specific needs. As a result, the services implemented by the DVCM vary over time, in keeping with the needs of current cluster applications.

In summary, host nodes use the DVCM's services to efficiently implement end user information services. The DVCM's functionality resides on the NIs and is accessed from host CPUs via communication instructions. The DVCM is extended and specialized much like extensible OS kernels developed for high performance server systems like SPIN and Exokernel[4, 9]. The architecture of DVCM is described in more detail in Section 2. Next, we present some of the specific DVCM extensions implemented and evaluated in this paper.

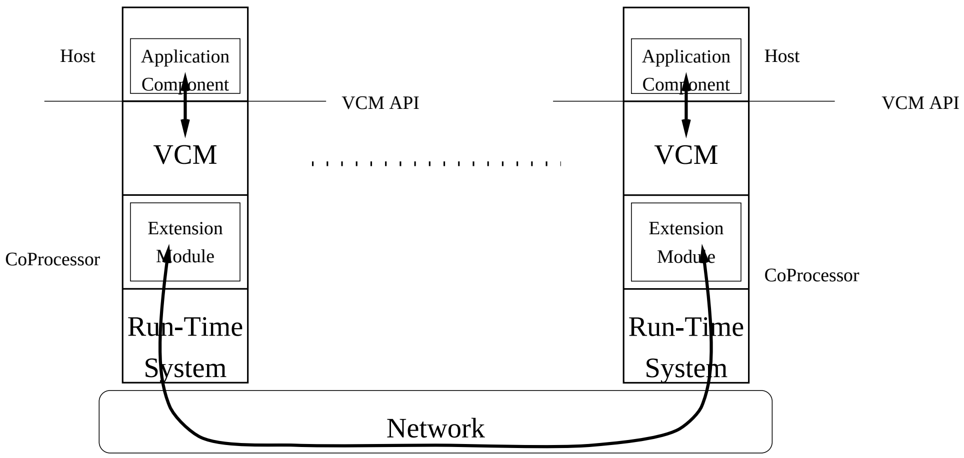

Figure 2: Distributed VCM Architecture.

Contributions. The DVCM idea, its realization for CoProcessor-based NI architectures, and its utility for attaining high performance on cluster machines have been explained and evaluated in previous papers[22]. This paper's novel contributions are the following:

- We demonstrate the feasibility of implementing the DVCM architecture on COTS (Commercial Off-The-Shelf) runtime support resident on NIs, namely, the VxWorks embedded real-time OS from Wind River Systems[34] running on Intel i960RD I2O boards. We show that performance-critical communication extensions can be executed with high performance on such standard CoProcessor platforms.

- We demonstrate the utility of placing certain services onto NIs vs. hosts. Such placements can improve performance in several ways. First, the service's *device-* or *network-near* functions typically run faster on the CoProcessor, because their execution does not involve I/O busses, host memory, and host CPUs. Second, on the host CPU, the time-critical execution of device interactions is easily jeopardized by the CPU's need to also run higher-level application services, and because the context switches typically required by such interactions are expensive due to the CPU's deep cache hierarchy and due to cache pollution. Third, a service running on an NI like the I2O boards not only removes load from the CPU but more importantly, it *eliminates traffic* across both the host's I/O busses and the communication network, thereby freeing up these valuable resources for other server actions. This is discussed in more detail next. Also, services like media schedulers running on host-CPUs are easily affected even by transient loading conditions whereas media schedulers running directly on I2O NIs are immune to host-CPU loading.

The traffic elimination we demonstrate for media applications utilizes window- and time-constrained scheduling techniques[33, 32]. Specifically, when a server streams video to some number of clients, packet scheduling is performed in order to guarantee differential packet rates and deadlines to meet clients' individual QoS needs. Packet scheduling eliminates traffic by implementing stream-selective lossiness in overload conditions. By performing packet scheduling on the NI rather than the host, traffic is eliminated for the media streams emanating from the disks attached to it (and to the host), thereby offloading the server's I/O busses, CPU, and memory resources. Furthermore, for distributed implementations of media streams on the cluster server, traffic elimination also occurs for media streams entering the NI from the network linking it to other cluster nodes. Also, packet schedulers running directly on the NI are unaffected by host loading unlike host-CPU based packet schedulers that easily get affected by transient loading.

In addition, the load imposed on host CPU, memory, and I/O bus is reduced due to selective traffic elimination on the NI. The scheduling overhead of the host-based DWCS (Dynamic Window Constrained Scheduling) scheduler as reported by us in [33, 32] is of the order of $\approx 50\mu s$. This result was obtained on an Ultra Sparc CPU (300 MHz) with quiescent load. The scheduling overhead of the i960 RD I2O card (66 MHz) based scheduler is around $\approx 65\mu s$. These results are comparable, although the i960 RD is a much slower processor (factor of 4).

Section 2 describes the architecture of the DVCM. Network CoProcessor based media scheduling is described in Section 3 with algorithm, design and construction of the DWCS embedded scheduler. Section 4 experimentally evaluates the embedded scheduler we have built and provides microbenchmarks, demonstration of streaming capabilities, impact of server load, comparison with an equivalent host-based scheduler and discussion of results. Related work is described in Section 5 and Section 6 concludes the paper.

2 DVCM Architecture

The architecture of the DVCM is comprised of three sets of functions (see Figure 2). The first set implements the DVCM's API, which gives each node's application programs efficient access to DVCM instructions. This communication-centric API is supported by extensions of the operating system kernel resident on each node. In the implementation described in this paper, these extensions are implemented as device drivers interacting with the I2O boards via PCI interfaces. In a previous implementation, a SUN Solaris kernel extension was used to implement DVCM for FORE ATM boards linking a distributed SUN Sparc-based cluster machine[22, 25]. In either case, the DVCM appears to the application program as a memory-mapped device, offering certain instructions, controlled via control registers, and sharing selected memory pages with the application, the latter containing the communication buffers utilized by these programs.

The second set of DVCM functions comprises low-level runtime support on the NI, used for implementation of DVCM instructions. The I2O-based implementation of DVCM as described in this paper, utilizes an embedded system configuration of the VxWorks real-time operating system[34] offering support for memory management, task creation, deletion, and scheduling, and device access for the network links and SCSI interfaces resident on the NI. The following functionality has been added to VxWorks in order to exploit this NI's specific hardware and to implement the cluster-wide actions of DVCM: fixed-point library to implement operations cheaply, driver front-ends to initialize controllers/storage, timestamp counter rollover management, circular queues and heaps.

The third set of DVCM functions are the extensions that support specific applications' needs. An example of this would be a scheduler for streaming media provided as an application specific extension by the DVCM and resident on the i960 RD I2O cards. Section 3.1 describes an application-specific extension namely, a scalable scheduler for media streaming (Figure 2).

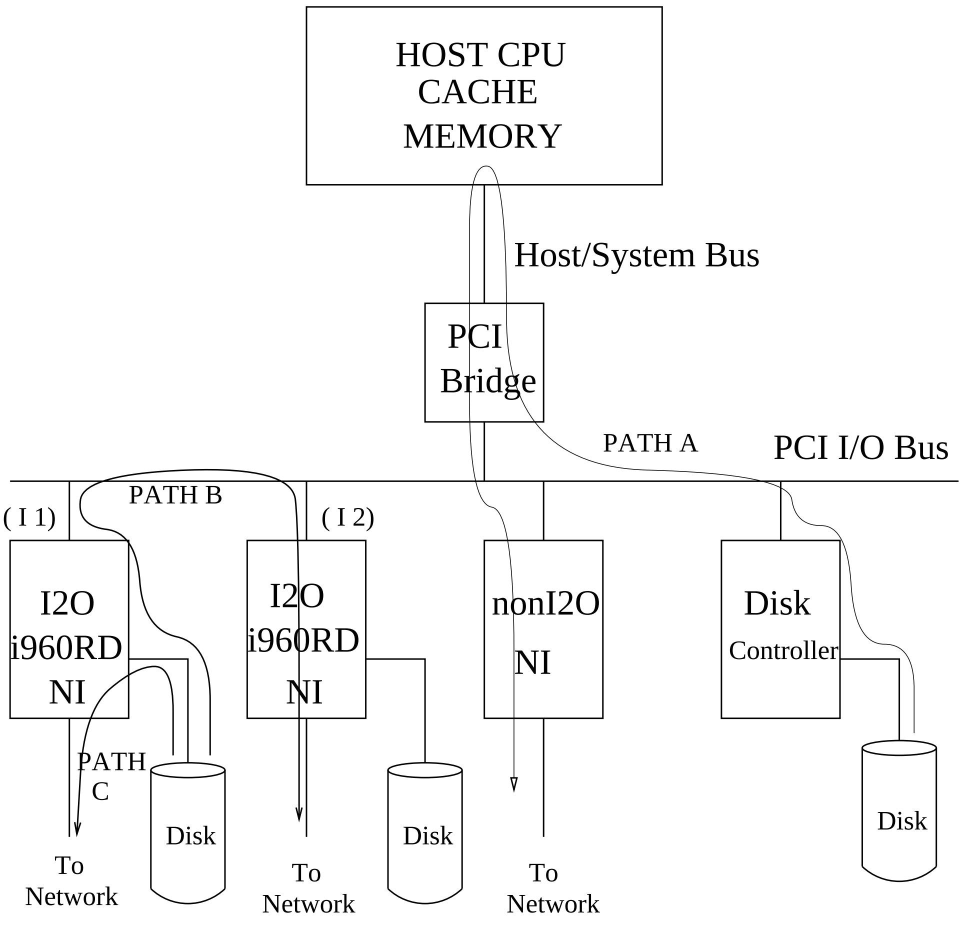

Figure 3: Frame Transfer Paths.

3 Network Co-processor Based Scalable Media Scheduling

The DVCM architecture allows run-time extensions for inclusion of services required for streaming media. This Section describes a service implemented by us for use in scheduling media streams. The algorithm, architecture and implementation issues are described in this Section.

3.1 Stream Scheduler Architecture

The NI-resident media scheduler has been implemented as an application-specific extension of the DVCM. Application threads on the host CPU that need to provide media streams requested by clients may use the DVCM application- specific media scheduler extension resident on the i960 RD I2O card. For purposes of this paper, the unit of streaming media is an MPEG-I frame. MPEG-I frame producers, running as application threads, may stream frames to remote clients using the NI-resident scheduler. The unit of scheduling for the NI-resident scheduler is also an MPEG-I frame. Remote consumers may forward frames to other consumers or buffer frames for display or storage, scheduled for delivery by the DVCM extension.

A scalable implementation of the DWCS scheduler may be realized by suitable structuring of software and distribution of components between the host CPU and NI CoProcessor. As stream requests to a server are increased, then the server must be able to process these requests with a pre-negotiated bound on service degradation expressed in terms of media scheduling parameters like delay-jitter, loss and throughput. A scalable implementation of the DWCS media scheduler must take into account the following [35, 7]:

- Performance impact of scheduling and memory hierarchies traversed by frames with the media scheduler running on the host or directly on the NI.
- Host CPU involvement during frame transfers from a media storage unit to an NI.
- Bus-domain traversals (system bus to PCI bus and vice-versa).

The media scheduler is based on the DWCS (Dynamic Window-Constrained Scheduling) packet scheduler described in [33, 32], where the scalability of the DWCS scheduler is demonstrated with respect to streaming media frames. One possible scalable target architecture is shown in Figure 3. Multiple NIs (in this case i960 RD I2O cards) are present on the I/O bus. One or more NIs may run the media scheduler and also have disks attached. The Figure 3 shows three paths - A, B and C. Path A represents the transfer of frames from a disk attached to a SCSI controller card to a separate non-I2O NI. An application thread on the host CPU that has opened the MPEG file for reading must transfer frames from the disk to the filesystem buffer cache (with possibly major portions in host CPU memory). The first part of this transfer involves traversal of memory hierarchies and bus domains (I/O bus to system bus). The second part of path A involves transfers of frames from host CPU memory to the network via the NI with suitable protocol encapsulation. Again, this path involves traversal of memory hierarchies and bus domains. Path B shows the transfer of frames from disks attached to an i960 RD card (I1) to another i960 RD media scheduler card (I2). This path only involves the I/O bus and completely eliminates host CPU or memory. Path C involves transfer of frames from a disk attached to an i960 RD network card to media scheduler input queues resident on the same card. Here, the path eliminates the I/O bus, host CPU and memory. Also, the media scheduler executes on the i960 RD card and does not consume host CPU cycles. Being closer to the network, the scheduler may be reconfigured based on network condition parameters. This again will not require message traversals across the I/O bus.

3.1.1 Design Considerations

A typical architecture of a system with NI-based scheduling is shown in Figure 1. One or more NIs in a system(compute node) may be dedicated to running the NI-based scheduler and other disk-attached NIs may serve as stream producers. The DWCS scheduler code module is embedded in the i960 RD I2O NI with the bootable system image of the VxWorks Operating System, usually resident on flash-ROM on the i960 RD I2O NI card. On system boot, all devices on the PCI bus boot up, which, brings up the VxWorks Operating System on each NI. Initialization code in the kernel is used to spawn the scheduler thread with code image resident on flash-ROM.

The following must be considered during design of an embedded NI-based scheduler on i960 RD I2O cards:

- Compact data structures(scheduler attributes or scheduler frame descriptors) for packet schedule representation that minimize the use of NI memory. Copies of frames must be minimized to conserve NI memory.
- Extensible scheduler design decoupling scheduling analysis and schedule representation(data structures). This allows different data structures to be used for experimentation(FCFS circular buffers, sorted lists, heaps or calendar queues)[35] with different packet schedule representations. Packets in a given stream(at the same priority level) may be scheduled in arrival order(FCFS) or based on a service tag associated with each packet.

- Scheduling and dispatch may be performed asynchronously with respect to each other. Asynchronous scheduling and dispatch may require an additional dispatch queue, but allows scheduling decisions to be made at a higher rate. Coupling scheduling and dispatch allows a single data structure to hold frame descriptors(or attributes) and conserves memory. Also, packets do not suffer additional queuing delay and jitter in dispatch queues[33, 32].

- A high degree of concurrency must be allowed in scheduler operation. Packets in any stream must be enqueued in scheduler data structures concurrent with scheduling analysis and dispatch of other packets. Scalable scheduler operation with respect to number of packet streams, stream rates and 'tightness' of deadlines and loss-tolerance, may be achieved by varying the latency or frequency of scheduler action[33, 32].

3.1.2 Scheduler Construction

The embedded construction of the DWCS scheduler is shown in Figure 4. Frames or packets are stored in circular buffers on a per-stream basis. Head-of-line packets in each stream form loss-tolerance and deadline heaps and encode stream priority values. The scheduler must pick the stream with the lowest priority according to rules described in [33, 32]. The host-based scheduler implementation uses System V shared memory between processes[33, 32] while the embedded NI CoProcessor implementation of the scheduler is lightweight and uses VxWorks tasks (threads) with physically pinned memory for data sharing between tasks. Frame producers on the host CPU or other PCI NIs transfer frames to the NI-based scheduler. The i960 RD I2O NIs are equipped with 4MB of on-board installed memory and may be expanded to 36MB. To conserve memory, we maintain a single copy of frames in NI memory and allow scheduling analysis and dispatch to manipulate addresses of frames. Storing frames directly in NI memory (rather than in host memory) reduces the overall scheduling analysis and dispatch latency of a single frame, the jitter experienced by frames and also may reduce the mean queuing delay of frames in a given stream as the frames are directly available in local NI memory and do not require a 'pull' from host memory. Alternatively, frames may be buffered in host memory and the NI may 'pull' frames. This adds to overall scheduling latency of a frame(more time to dispatch), may cause increased delay-jitter and increase the mean queuing delay of frames.

As shown in Figure 4, a circular buffer is maintained for each stream with separate head and tail

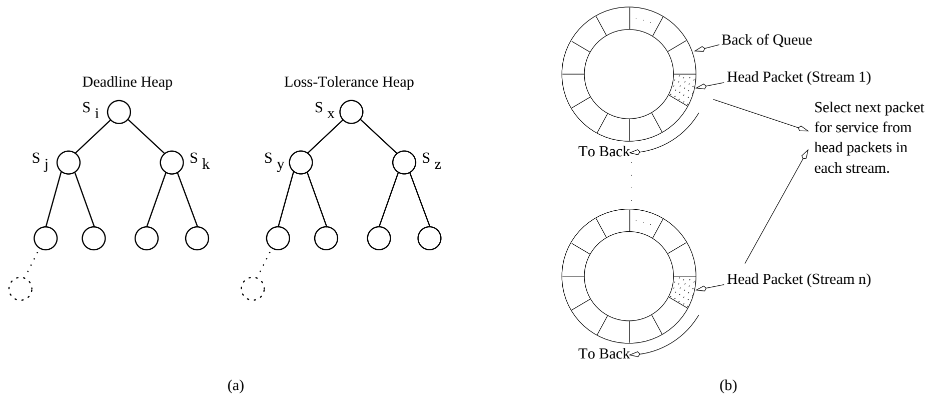

Figure 4: (a) This implementation of DWCS uses two heaps: one for deadlines and another for loss-tolerances.(b) Using a circular queue for each stream eliminates the need for synchronization between the scheduler that selects the next packet for service, and the server that queues packets to be scheduled.

pointers. Frame producers may inject frames into the scheduler using the tail pointer and the scheduler may read frames using the head pointer. This eliminates any synchronization needs between readers and writers.

DWCS is designed to maximize network bandwidth usage in the presence of multiple packets each with their own delay constraints and loss-tolerances. The per-packet delay and loss tolerances must be provided as attributes after generating them from higher-level application constraints. The algorithm requires two attributes per packet, as follows:

- *Deadline* – this is the latest time a packet can *commence* service. The deadline is determined from a specification of the maximum allowable time between servicing consecutive packets in the same stream.
- *Loss-tolerance* – this is specified as a value x_i/y_i, where x_i is the number of packets that can be lost or transmitted late for every *window*, y_i, of consecutive packet arrivals in the same stream, i. For every y_i packet arrivals in stream i, a minimum of $y_i - x_i$ packets must be scheduled on time, while at most x_i packets can miss their deadlines and be either dropped or transmitted late, depending on whether or not the attribute-based QoS for the stream allows some packets to be lost.

At any time, all packets in the same stream have the same loss-tolerance, while each successive packet in a stream has a deadline that is offset by a fixed amount from its predecessor. Using these attributes, DWCS: (1) can limit the number of late packets over finite numbers of consecutive packets in losstolerant or delay-constrained, heterogeneous traffic streams, (2) does not require a-priori knowledge of the worst-case loading from multiple streams to establish the necessary bandwidth allocations to meet per-stream delay and loss-constraints, (3) can safely drop late packets in lossy streams without unnecessarily transmitting them, thereby avoiding unnecessary bandwidth consumption, and (4) can exhibit both fairness and unfairness properties when necessary. A detailed description of the DWCS scheduler is provided in [33, 32].

4 Experimental Evaluation

This section presents an experimental evaluation of the system presented in Section 3.1. We demonstrate that offloading the media scheduler from the host CPU to the NI is feasible and efficient because of fewer overheads and *network-near* operation not involving host/PCI bus domain traversals. Microbenchmarks have been constructed that measure important system primitives. The experiments in this section also demonstrate the functioning of a NI-based scheduler and compare it with a host-based scheduler. The experimental infrastructure will be described first, followed by the experiments.

4.1 Experimental Setup

The experimental setup consists of a Quad Pentium Pro server(4 X 200Mhz) running Solaris 2.5.1 X86 with 128 MB of memory. Three I2O cards are placed in separate PCI slots on the same bus segment. One I2O card hosts the scheduler and scheduler data structures. Here, the packet scheduling functionality on the host has been offloaded to one of the i960 RD I2O cards allowing *network-near* operation. The other two cards serve as stream sources for MPEG traffic. Disks are directly attached to the I2O cards that store MPEG files used to source streams. An MPEG segmentation program developed in [33, 32] is used for segmenting an MPEG encoded file into I, P and B frames and serves as a stream producer. This emulates the MPEG file segmentation process in an MPEG player. The MPEG segmentation process may be run on the host CPUs or the I2O cards. MPEG stream producer processes or threads on the host or I2O cards inject frames into the scheduler queues on the scheduler I2O card using PCI bus transfers. The scheduler then picks frames based on scheduling criteria and dispatches frames on the network. Client machines running MPEG players may attach to the scheduler card for MPEG stream delivery. We have built mechanisms to measure desired performance parameters at the scheduler card or at the remote client end. Remote client machines connect to the scheduler card using a 100Mbps ethernet switched interconnect [33, 32, 13, 18, 34].

4.2 Microbenchmarks

The following benchmarks have been constructed to show that off-loading scheduling from the host to the NI is feasible. The experiments described above show that scheduler functionality may be off-loaded to i960 RD NIs with a scheduling overhead of $\approx 65\mu$s. This corresponds to around half an Ethernet frame time ($\approx 120\mu$s) on a 100Mbps link. Also, the output of the scheduler is *network-near* at the exit point to the network and does not require traversals across the I/O bus to the network. Microbenchmarks have been developed that measure system primitive performance. For the purposes of the microbenchmarks, we start the scheduler after all frame descriptors have been written into the circular buffer (we store addresses of frame descriptors in the circular buffer). 'Total Sched Time' records the time to schedule all the frames on the network. 'Avg Frame Sched Time' records the average time to schedule a single frame on the network. Similarly 'Total time w/o Scheduler' measures the cumulative time to transmit all the frames on the network without the scheduler. We simply re-route execution in the code to a point where the address of the frame to be dispatched is readily available and does not need scheduler rules. Similarly 'Avg Frame Time w/o Sched' records the average time to transmit a single frame without the scheduler. Table 1 records microbenchmarks for both a software floating point version and a fixed-point version. The i960 RD is an I/O CoProcessor and does not have a floating point unit. Wind River Systems (see [34]) has provided a software floating point(FP) library that may be configured into the VxWorks kernel. Some of the scheduler computations involve floating point operations and we have tried to limit the overhead of using the software floating point library by developing our own fixed-point version (arguments are simply stored as fractions with numerator and denominator with divisions implemented as shifts). The overhead of using the VxWorks software FP library is around $\approx 20\mu$s. The results from Table 1 show a scheduler overhead (difference between Average Sched time with scheduler and Average Sched time without the scheduler) of $\approx 75\mu$s for the fixed point version with data cache disabled.

Table 2 records the above parameters with data cache enabled. The VxWorks driver we have used currently supports disk accesses with data cache disabled. For measurements in Table 2, we read the MPEG file from disk and populate the circular buffer (the disk driver disables the data cache automati-

Microbenchmark	Software FP (μSecs)	Fixed Point (μSecs)
Total Sched time	19580.88	16425.36
Avg frame Sched time	129.67	108.48
Total time w/o Scheduler	5210.88	4583.28
Avg frame time w/o Scheduler	34.6	30.35

Table 1: Scheduler Microbenchmarks(Data Cache Disabled)

cally on reboot). After this we enable the data cache, since further accesses to the locally attached disk are not required. Table 2 shows that the presence of the data cache improves average frame scheduling time for both software FP and Fixed point implementation by $\approx 14.47\mu$s and $\approx 13.88\mu$s respectively. For streaming media, we do not expect data caching to help considerably. The microbenchmarks in Table 1 and Table 2 however, show the benefit of possible data caching of scheduler data structures. As scheduling decisions are made on a frame-by-frame basis, data caching has the effect of reducing the total Scheduling time by $\approx 2182.32\mu$s and $\approx 2129.76\mu$s for the software FP and fixed-point implementation respectively. The scheduler overhead as reported in [33, 32] for the host-based DWCS scheduler is $\approx 50\mu$s. Results from Table 2 show a scheduler overhead of $\approx 66.82\mu$s (difference between Average Sched time with scheduler and Average Sched time without the scheduler) for the fixed point version.

Microbenchmark	Software FP (μSecs)	Fixed Point (μSecs)
Total Sched time	17398.56	14295.60
Avg frame Sched time	115.20	94.60
Total time w/o Scheduler	4776.48	4195.68
Avg frame time w/o Scheduler	31.40	27.78

Table 2: Scheduler Microbenchmarks(Data Cache Enabled)

The fixed point version of the scheduler improves the scheduling decision time by around $\approx 20\mu$s. The scheduler operations require fractional values to one or two decimal places (implemented easily with a structure representing a fraction). The VxWorks software FP library simply eases the development process by allowing float datatypes in the code and other operations as if the code were being run on a processor with a floating-point unit. Using the fixed point version does not affect the quality of scheduling (expressed in terms of parameters like delay-jitter, loss and throughput) as a structure representing a fraction can easily capture the accuracy and numerical stability required of the operations in the scheduler. The floating-point version simply provides a convenient way of implementing the operations in the scheduler(at the cost of performance). The presence of the data cache has helped improve the scheduling decision time and the total scheduling time of the stream. This improves scheduler computation time per frame by $\approx 15\mu$s. Enabling the data cache allows stream priority values and descriptor addresses to be cached and updated every scheduler cycle without additional memory latency. In our experimental setup, multiple NIs may reside on the PCI bus and one or more NIs may host the scheduler thread and corresponding data structures. Stream producers are run separately on NIs with disk attached and provide frames to Scheduler-NIs (exclusively running the scheduler thread, with no disks attached allowing data caching) across the PCI bus. This allows the scheduler thread to exploit the benefits of data caching without being limited by the disk driver disabling the data cache [33, 32, 13, 18, 34].

4.2.1 'Hardware Queue' Implementation

The i960 RD I2O cards provide a number of hardware resources for I2O device operation. These include outbound and inbound circular queues and index registers. The 'Hardware Queues' on the i960 RD I2O card are a set of 1004 32 bit memory-mapped registers in local card address space. Accesses to the memory-mapped registers do not generate any external bus cycles(off-processor core). The set of benchmarks described in Table 3 are an attempt to investigate if indexing into a circular buffer of frame descriptors may be done faster if they reside in memory-mapped register space. We have implemented a circular buffer using these registers. Each 32 bit register holds the descriptor(with address and other attributes) of an MPEG frame. Table 3 lists the primitives recorded in Table 1 and 2 with measurements using 'Hardware Queues' with data cache enabled and for the faster fixed-point version.

Microbenchmark	Fixed Point (μSecs)
Total Sched time	14569.68
Avg frame Sched time	72.48, 96.48
Total time w/o Scheduler	4199.04
Avg frame time w/o Scheduler	27.80

Table 3: Scheduler Microbenchmarks(Data Cache Enabled)

The results in Table 3 are comparable to the results in Table 2. For the results in Table 2, the frame descriptors are stored in memory(all memory pages are pinned in our configuration of VxWorks). Frame descriptors hold the frame addresses and other attributes of frames. The scheduler loops through the frame descriptors and picks the eligible descriptor. All addresses of frame descriptors are in local card memory address space and do not generate any external PCI bus cycles. Similarly, for the results in Table 3, the frame descriptors are stored in local memory mapped register space. The cost of looping through descriptors in local memory-mapped register space or in pinned memory pages for the i960 RD appears to be comparable. In both implementations, it must be noted that the actual frames are located in pinned local memory address space, the descriptors are however stored in memory-mapped register space or pinned local card address space.

Offloading Scheduler functionality from the host to the NI is feasible and because card-to-card PCI transfers may be completed without host CPU involvement, is advantageous. This may lead to increased CPU utilization on the host as the host CPU is free to do other tasks [33, 32, 13, 18, 34].

4.2.2 Media streaming from NI attached disks

A host CPU based scheduler as described by us in [33, 32] uses host filesystem buffers for frames, I/O and system bandwidth, host memory and kernel/user space buffers for scheduling frames on the network. An MPEG file resident on disk, must be transferred to host filesystem buffers by the disk controller card on the I/O bus. This is shown as path A in Figure 3. An NI on the I/O bus may however use PCI card-to-card transfers for transfer between disk controllers and scheduler input queues completely eliminating the use of any host based resources or the host system bus. This is shown as path B in Figure 3 . The i960RD I2O card consists of two SCSI ports and two 100 Mbps Ethernet ports as shown in Figure 1. Disks may be attached to the SCSI ports and media may be streamed directly through to the network using the 100 Mbps ethernet port. The host CPU, I/O bus and host CPU filesystem are completely eliminated from the transfer path. This is shown as path C in Figure 3. A more scalable way to stream media from storage disks to the network without host CPU/filesystem involvement is to attach disks to a separate i960 RD card and transfer frames from disk across the PCI bus to a separate Scheduler-NI that schedules frames on the network. This is shown in Figure 3 as path B. This Section presents results from critical path benchmarks recorded for three different configurations of frame transfers. All benchmarks measure the latency of a 1000 byte frame transfer from disk to remote client (over an Ethernet network) averaged over 1000 transfers. The measurements in Table 4 record the latency of a single frame transfer. Consider Experiment I in Table 4. This is similar to Path A in Figure 3. An MPEG file on internal system disk attached to a disk controller on the PCI bus is streamed to a remote client. The second experiment (Path C in Figure 3) consists of a single i960 RD NI with a disk attached. Bus activity is reduced to a minimum by disabling other cards on the same bus segment. A thread running on the i960 RD NI reads frames from the locally attached disk and serves frames to a remote client attached to the NI over an Ethernet link. The third experiment III (Path B in Figure 3) consists of two NI cards. This path involves transfer of frames from disk (attached to an NI or separate disk controller) over the PCI I/O bus (using PCI peer-peer DMA transfer) to a separate i960 RD I2O NI which transfers frames to the remote client across the network. This transfer does not involve consumption of host memory, host CPU cycles or host system bus bandwidth.

Expt	Frame Transfer Path (1000 byte frame)	Frame Transfer Time (msec)
I	Disk-Host CPU-I/O Bus-Network (no load w/ cache)	1(ufs)/8(VxWorks)
II	NI Disk-NI CPU-Network (no cache)	5.4
III	Disk-I/O Bus-NI CPU-Network (no cache)	5.415 (4.2disk+1.2net+0.015pci)

Table 4: Critical Path Benchmarks

The latency component common to Experiment I, II and III is the disk access time which is $\approx 4.2ms$ for a single frame (4.2disk in Table 4). Transfer time to the remote client is $\approx 1ms$ for a single frame (1.2net in Table 4, data cache disabled). This represents the latency from end-to-end including traversal of network stacks at either end and wire transmission time. Also, transfer time from I2O NI card to I2O NI card across the PCI bus is $\approx 15\mu s$ for a single frame (we used DMA writes from card-to-card to achieve this). Table 5 summarizes the results from benchmarking transfers on the PCI bus using DMAs and PIO reads/writes. The results in Experiment I show a frame latency time of $\approx 8ms$ when the VxWorks filesystem was mounted on Solaris. These were obtained on a Solaris 2.5.1 system using an Intel 82557-based NI . The system disk attached to the system disk controller was used to serve frames using the Intel NI. The VxWorks filesystem is a dos-based filesystem and this was mounted on the Solaris host.

To mitigate the effects of filesystem and disk layout performance variation, the same filesystem and disk was used with Experiment I, II and III.

It is interesting to note that when the Solaris UFS filesystem is mounted (for Experiment I) and MPEG files are stored on disk, the frame latency time is only 1 ms. UFS uses a logical block size of 8K, may cache and prefetch blocks for better performance. VxWorks does not support UFS filesystems and we were unable to mount UFS with VxWorks for Experiments II and III[8, 18].

Media stream transfer directly from NI-attached disks to remote clients is advantageous because these transfers may occur with lower overhead than streams being transferred from host system disks under the control of the host CPU and host OS. This as shown in Table 4 can effectively bring down the frame transfer latency time significantly. Note the advantage of transferring from disk-attached NI to a

Benchmark	Time (μSecs) / BW (MB/s)
MPEG File Transfer by DMA(773665 bytes)	11673.84 / 66.27
Memory Word Read(PIO)	3.6
Memory Word Write(PIO)	3.1

Table 5: PCI Card-to-Card Transfer Benchmarks

dedicated scheduler-NI as in Experiment III(cf. Experiment II). The difference is $\approx 0.015ms$ incurred on account of PCI arbitration and possible synchronization costs in frame send/receive queues. The configuration in Experiment III allows an NI to be dedicated to Scheduling frames and avoids NI-CPU contention between frame scheduling/dispatch and frame production. This is done by relocating the frame producers to a separate disk-attached NI. This can help in effectively balancing the stream transfer load between different NIs on the I/O bus. Inter-card (peer-peer) transfers do not involve the host CPU or memory subsystem and can increase host CPU utilization for other tasks by traffic elimination.

4.2.3 Comparison of Host- and NI-Based Scheduling

A packet scheduler running on the host, may schedule frames on the network along path A in Figure 3. Frame transfer to scheduler input queues, scheduling and dispatch are the three main actions involved in host-based scheduling. These activities involve consumption of host filesystem buffers, system bus bandwidth, I/O bus bandwidth, host memory and kernel/user space buffer usage. Depending on the frame transfer path to scheduler input queues (path B or path C in Figure 3), NI-based schedulers may completely avoid consumption of I/O bus bandwidth (path C in Figure 3), host memory and host bus bandwidth[13, 18, 34] and are unaffected by host CPU load. A host-based scheduler is easily affected by the host Operating System's need to run higher-level application services (even a minimal installation runs system daemons; database and cluster-connectivity enabled machines run additional services). With a large number of tasks competing for the host CPU, effects of CPU contention on the host-based scheduler may be seen as degradation in scheduling quality (expressed as decrease in output bandwidth available for a stream and increase in frame queuing delay).

The (frame/packet) scheduler receives CPU at lower rates because of increased service load leading to back-logged frames in scheduler input queues that result in missed deadlines and loss-tolerance violations. This may trigger packet-dropping leading to lower scheduling quality. For jitter-sensitive traffic, variation in the rate at which the (frame/packet) scheduler receives CPU may increase delay-jitter already experienced by frames. Some frames in the same stream may be serviced at more frequent intervals by the (frame/packet) scheduler and others at less frequent intervals leading to jitter in frame inter-arrival times. Also, frames may need to traverse multiple bus/network scheduling domains (system bus to I/O bus and then to the network) and memory hierarchies causing variation in frame inter-arrival times, leading to increased delay-jitter[33, 32, 13, 18, 34]. An NI-based scheduler runs on the NI Operating System(VxWorks in our case). A stand-alone embedded VxWorks configuration may run few system tasks(threads) scheduled by the native 'wind' scheduler. The NI Operating System is dedicated to running the (packet/frame) scheduler and needed network services. Frame Producers(not running on the scheduler-NI, DMA frames to scheduler input queues) do not affect the rate at which the (frame/packet) scheduler receives NI-CPU. The (frame/packet) scheduler on the NI may thus, receive NI-CPU at a rate with lower variability unlike the action of the host-based scheduler described above. Frames are serviced at a rate with lower variability eliminating excessive back-logging in queues. This may result in lower deadlines missed, lower loss-tolerance violations and more uniform delay-jitter variation. As frames on the NI may be output to the network directly without traversal of bus domains and memory hierarchies, jitter-sensitive traffic may experience more uniform jitter-delay variation.

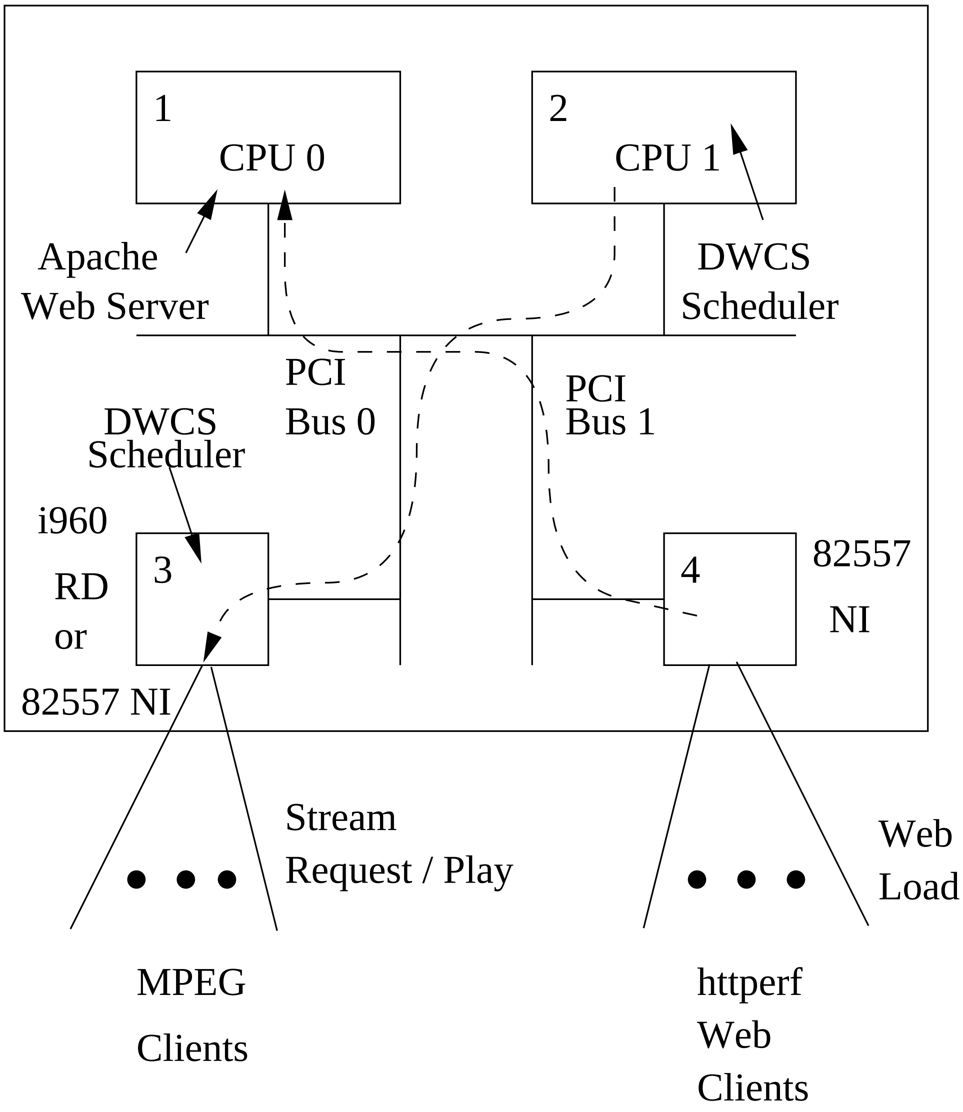

Figure 5: Server Loading Architecture - Web and Media Traffic.

Impact of Server Load on Scheduling. The experimental setup consists of a Quad Pentium Pro server (4 X 200Mhz) running Solaris 2.7 X86 with 128 MB of memory. This machine has two separate PCI bus segments and we place NIs on each of the bus PCI segments as shown in Figure 5.

For host-based scheduling load experiments, Intel 82557 100Mbps transceiver based NIs are placed in separate slots on each of the two bus segments (components 3 and 4 in Figure 5). For NI-based scheduling load experiments, one of the Intel 82557 NIs is replaced with a i960 RD I2O NI (component 3 in Figure

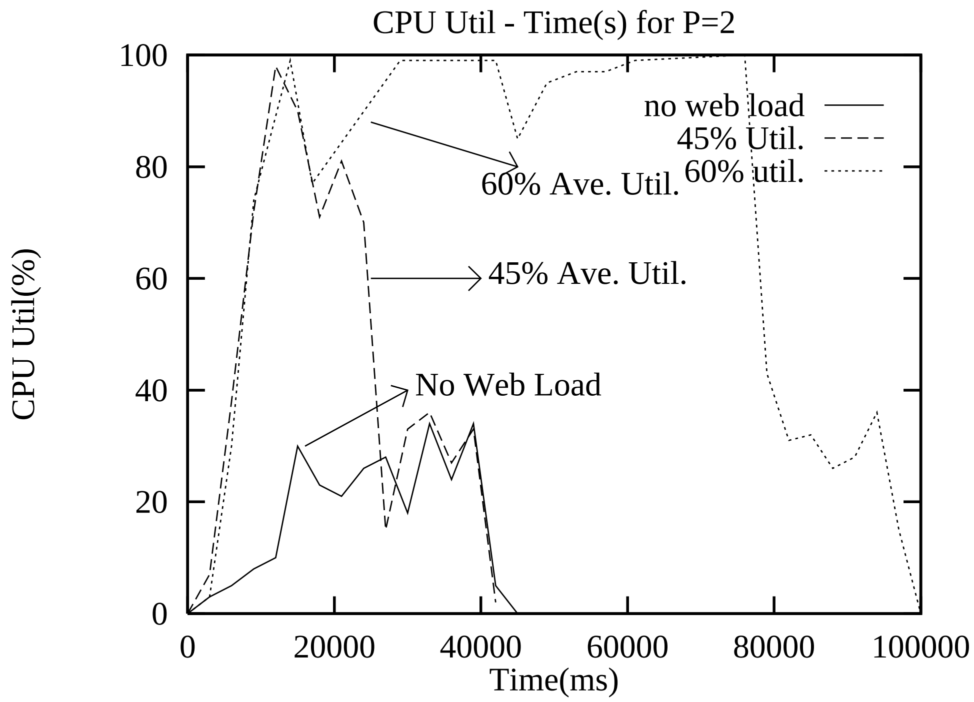

Figure 6: CPU Utilization Variation with Server Load.

5). The machine runs the Apache web server version 1.3.12 (with a maximum of 10 server processes and starting process pool with five server processes)[3]. The web server is loaded using 'httperf' (version 0.6)[17] from remote Linux-based clients. Flexible specification of load from remote clients is allowed by 'httperf' - web pages may be requested at a certain rate by a number of connections with a user-specified ceiling on the total number of calls. The experimental infrastructure is shown in Figure 5. Availability of two separate NIs on separate bus segments allows separation of web load and stream traffic. One of the NIs (with IP address bound to the Intel 82557-based NI) is used to load the web server using 'httperf' client traffic while, the other NI (with a different IP address bound to the NI, 82557-based or i960 RD I2O NI) is used to request and source stream traffic.

The first set of experiments involves the host-based scheduler version of the DWCS algorithm as described in [33] for a study of system loading effects on scheduler performance. Here, the DWCS scheduler runs completely on the host-CPU and data structures are maintained in host-CPU memory. For these experiments, two of the CPUs are brought off-line for a total of two on-line CPUs. The Apache web server in a configuration described above is brought up and bound to an IP address (of one of the

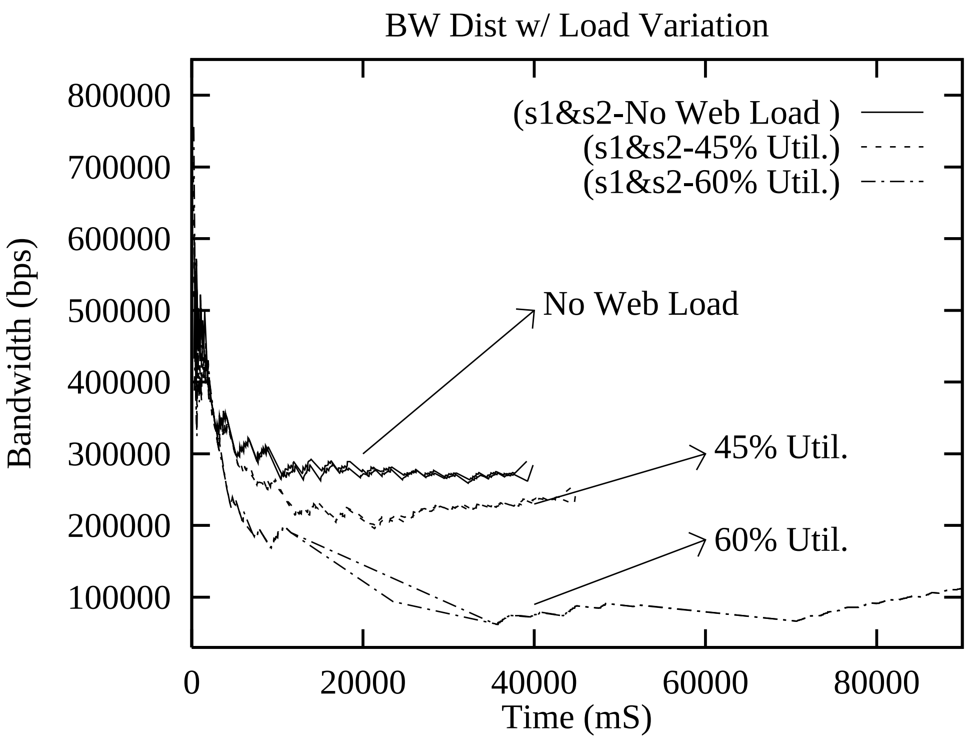

Figure 7: Bandwidth Variation with Load.

NIs). The DWCS scheduler is initiated and bound to one of the processors (using the 'pbind' Solaris facility)[8] with client requests accepted on a separate IP address bound to a different NI . This allows 'httperf' web clients to connect and load the Apache Web server using a specific IP address (bound to a specific NI). Similarly, MPEG clients may connect to a different IP address (bound to a different NI) for stream delivery. This experiment involves components 1, 2, 3 and 4 as shown in Figure 5 with component 3 as an Intel 82557 NI. Two MPEG clients shown as streams s1 and s2 connect to the system.

Figure 6 shows the total CPU utilization (measured using Solaris Perfmeter)[8] when the host-based scheduler is run without any load imposed by the remote web clients, a peak of around 35% is seen with an average utilization of 15%. Corresponding bandwidth variation and mean queuing delay of frames is shown for two streams s1 and s2 in Figure 7 and 8.

The system is then loaded using the remote web clients as shown in Figure 6 and stream requests are made to the scheduler simultaneously. Figure 7 and Figure 8 represent the corresponding bandwidth and queuing delay variations for the same two streams. The utilization represents the total utilization including CPU utilization because of streaming to clients. Load from web clients is applied at two levels

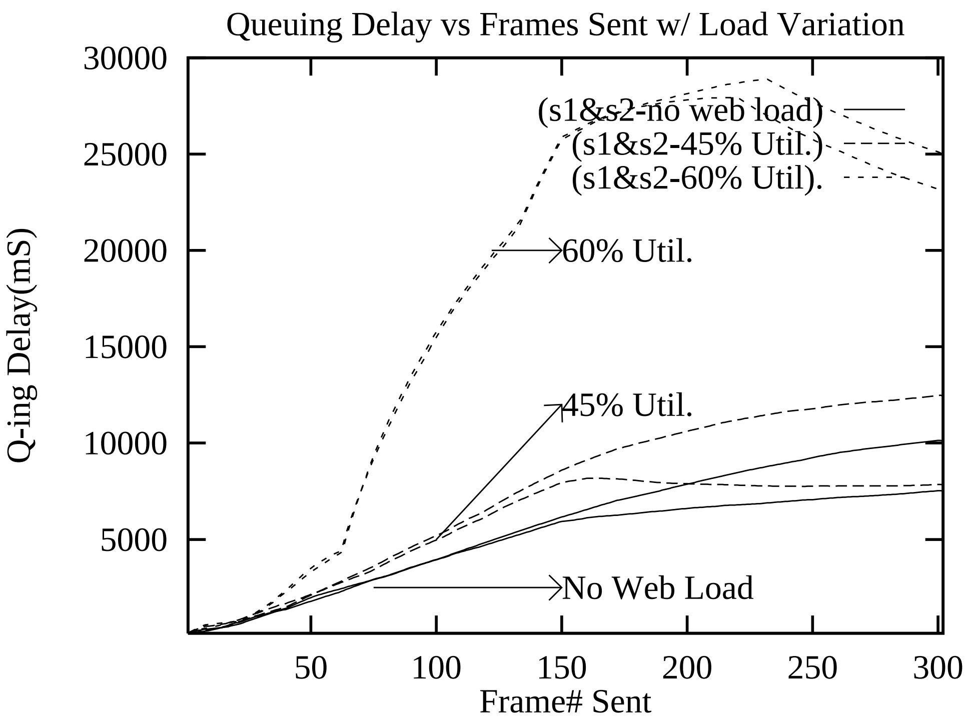

Figure 8: Queuing Delay Variation with Load.

- at the 45% utilization level and the other at the 60% utilization level (in two separate experiment sets as seen in Figure 6). For the 45% average utilization load, a decrease in bandwidth to 200,000 bps is seen at the 15s-20s time mark in the presence of load and the bandwidth can settle to only 230,000 bps (see Figure 7). The queuing delay graph in Figure 8 also shows the effects of loading, frames suffer additional queuing delay of around 2s in the presence of load. For the 60% average utilization case, severe degradation is seen. The bandwidth variation in Figure 7 shows a decrease in bandwidth to around 100,000 bps when the CPU utilization is in the excess of 80% (see Figure 6) during the period from 40s-80s. The bandwidth settles to less than 125,000 bps - half of the bandwidth seen in the absence of web server load (see Figure 7). Frames experience excessive queuing delay in the presence of load (60% average utilization), upto three times (30,000 ms) that seen in the absence of load (10,000 ms).

The next set of experiments involves the NI-based scheduler. For purposes of this experiment, one CPU is brought off-line (for a total of one on-line CPU) with one 82557-based Intel NI used for web server loading and a i960 RD based I2O NI used for MPEG streaming (DWCS runs on the NI) . This experiment involves components 1, 3 and 4 with component 3 as an i960 RD I2O NI which can stream to clients directly. Initially, streams are played to MPEG clients in the absence of any web server load with bandwidth variations and queuing delay measured on the NI-CPU. Next, the system is loaded using the load profile shown in Figure 6 (for 60% average utilization). The i960 RD I2O NI is placed on a separate bus segment and MPEG frames are streamed to clients by the NI-based scheduler. Loading of the web server is done using the other NI placed on a separate bus segment.

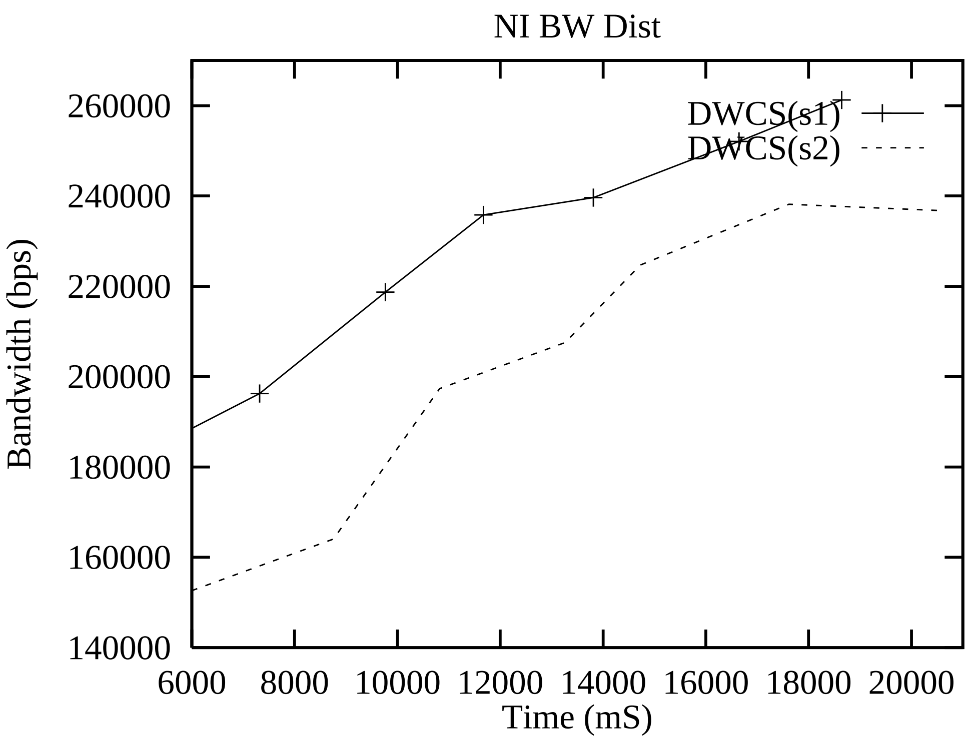

Figure 9: NI Bandwidth Distribution Snapshot: Unaffected by System Load.

The NI based scheduler is completely immune to web server loading and the bandwidth variation and queuing delay experienced by frames is shown in Figure 9 and 10 for both the cases of loaded and unloaded server (for streams s1 and s2). A settling bandwidth of around 260,000 bps is seen (for stream s1) which, is similar to the settling bandwidth achieved by the host-CPU based scheduler in the absence of load (250,000 bps in Figure 7). This represents traffic completely eliminated from the host system bus. A maximum queuing delay of 11,000 ms (stream s1) is seen (cf. 10,000 ms seen in the case of Figure 8).

Discussion of Results. The scheduling overhead of the host-based DWCS scheduler as reported by us in [33, 32] is of the order of $\approx 50\mu s$. This result was obtained on an Ultra Sparc CPU (300 MHz)

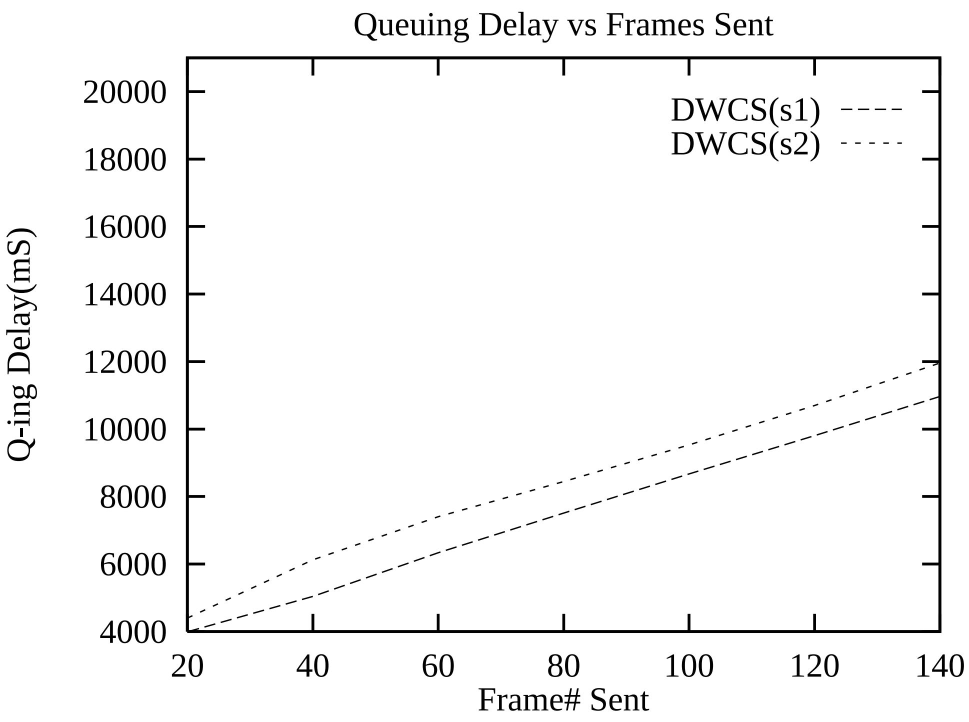

Figure 10: NI Queuing Delay Snapshot: Unaffected by System Load.

with quiescent load. The scheduling overhead of the i960 RD I2O card (66 MHz) based scheduler is around $\approx 65\mu$s. These results are comparable, although the i960 RD is a much slower processor (by a factor of 4). The results from Figure 7 and 9 above show that scheduling performance (in terms of bandwidth and queuing delay) is comparable between host-CPU based schedulers (in the absence of load) and NI-CPU based schedulers. This makes NIs attractive candidates for offloading stream scheduling from host-CPUs. The presence of even transient load adversely affects host-CPU based schedulers with performance degradation seen for CPU utilizations as low as 45% and severe degradation seen for CPU utilizations of 60% and higher. This shows that packet or frame scheduling is an activity that must be offloaded and embedded into the NI for enhanced streaming of multimedia. Inter-card transfers between peers on the same I/O bus segment enable traffic elimination from the host CPU and memory subsystem while allowing frame producers to transfer frames to schedulers.

5 Related Work

A number of NI-based research projects have focused on providing low-latency message passing over cluster interconnects like ATM, Myrinet, FDDI and HIPPI[10, 19, 29, 28] using intelligent NIs equipped with programmable CoProcessors[6, 13, 18, 22]. The network interfaces used in many cluster interconnects are intelligent and equipped with programmable co-processors[6, 13, 18, 26]. This makes it an attractive target to offload certain host tasks to allow tighter integration between computation and communication. Our DVCM communication machine implementation on FORE SBA-200 (i960CA) cards allows run-time extension of NI functionality and enables computation directly on the NI[22]. The SPINE project at the University of Washington involves construction of a custom OS for the NI with a focus on safety ([12]) and support for video and protocol processing on the NI. The project does not directly address the issue of video frame scheduling ([12]).

Recent research has put substantial effort into the development of efficient scheduling algorithms for media applications. Compared to such work, given the presence of some underlying bandwidth reservation scheme, the DWCS algorithm has the ability to share bandwidth among competing clients in strict proportion to their deadlines and loss-tolerances. This is a property shared with a number of recently developed algorithms. Fair scheduling algorithms[35, 20] (some of which are now implemented in hardware[21]) attempt to allocate $1/N$ of the available bandwidth among N streams or flows. Any idle time, due to one or more flows using less than its allocated bandwidth, is divided equally among the remaining flows. This concept generalizes to weighted fairness in which bandwidth must be allocated in proportion to the *weights* associated with individual flows.

The I2O industry consortium has defined a specification for development of I/O hardware and software. It allows portable device driver development by defining a message-passing protocol between the host and peer I/O devices[13, 18, 34]. The focus is on relieving the host from tasks that may be offloaded to a programmable NI. A number of efforts by industry include I2O cards for RAID storage sub-systems and off-loading TCP/IP protocol processing to the NI from the host[13, 18, 34].

There has also been a significant amount of research on the construction of scalable media servers. For example, recent work by Jones et al. concerns the construction and evaluation of a reservation-based CPU scheduler for media applications[15] such as the audio/video player used in their experiments. These results demonstrate the importance of explicit scheduling when attempting to meet the demands of media applications. If DWCS performed its scheduling actions using a reservation-based CPU scheduler like that described in [15], it would be able to closely couple its CPU-bound packet generation and scheduling actions with the packet transmission actions required for packet streams. Similarly, DWCS could also take advantage of the stripe-based disk and machine scheduling methods advocated by the Tiger video server, by using stripes as coarse-grain 'reservations' for which individual packets are scheduled to stay within the bounds defined by these reservations[7].

This work builds on the work described above to provide a unique combination of research to address the problem of media scheduling for scalability. - Extensible Virtual Communication Machine(DVCM) software on custom or commodity NI OS, which allows extensions for computations directly on the NI. DVCM includes an end-system scheduler(offloaded from host to provide *network-near* operation and tighter integration with the network with increased utilization for other tasks). In addition, commodity NI architecture (using I2O i960 RD) allows traffic elimination from host by transfers between peer I/O entities for scalability.

6 Conclusions and Future Work

We have built an embedded NI CoProcessor based scheduler with a focus on scalability using commodity hardware and software. The tradeoffs of storing descriptors in appropriate data structures for scalability and embedded construction is considered. The tradeoffs of using different hardware resources on the i960 RD I2O card is also considered. Our experiments indicate that the performance is comparable to an equivalent host-based scheduler. Computing packet scheduling decisions on the NI and integrating this tightly with the network is shown to have benefits. These may be realized in terms of increased host-CPU utilization by traffic elimination from the host bus and memory subsystems. Also, packet schedulers running on host-CPUs are easily affected even by transient loading conditions as demonstrated by us, whereas packet schedulers running directly on NIs are immune to host-CPU loading. Offloading frame/packet scheduling on the NI is a viable option.

The approach to scalability for media streams advocated by this paper is three-fold. The first is by scalable scheduler algorithm design demonstrated by us in [33, 32] and lightweight embedded construction shown in this paper. The second is by architectural distribution of resources(in this case i960 RD I2O cards, PCI bus segments and disks) to allow transfer between peer entities on the I/O bus eliminating host bus bandwidth and host memory consumption and third by offloading scheduling to NIs which can be immune to host-CPU transient load. Scalability for a large number of streams may require careful construction. Given the limited I/O slot real-estate, careful balance between NIs dedicated for scheduling and stream sourcing is required. **Future Work.** Experimentation is underway for studying bandwidth allocations for a large number of streams streamed by the scheduler. We are looking at ways of improving scheduling decision time using FPGAs(Field Programmable Gate Arrays) to augment CoProcessor functionality[31].

Acknowledgements. We would like to thank the reviewers for their many invaluable suggestions. Our thanks to other contributors to this research, including Prof. Sudhakar Yalamanchili, Sarat Manni and S. Roy.

References

[1] Kevin Almeroth and Mostafa Ammar. A scalable, interactive video-on-demand service using multicast communication. In *Proceedings of the International Conference on Computer Communication Networks*, San Franciso, California, September 1994.

[2] Thomas E. Anderson, David E. Culler, David A. Patterson, and the NOW Team. A Case for Networks of Workstations: NOW. *IEEE Micro*, Feb. 1995.

[3] Apache http Server Project Apache Software Foundation. http://www.apache.org/httpd.html.

[4] Brian N. Bershad, Stefan Savage, Prsemyslaw Pardyak, Emin Gum Sirer, Marc Fiuczynski, and Becker Eggers Chambers. Extensibility, safety, and performance in the SPIN operating system. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, pages 267–284, Copper Mountain, Colorado, December 1995.

[5] Ben Blake. *A Fast, Efficient Scheduling Framework for Parallel Computing Systems.* PhD thesis, Department of Computer and Information Science, The Ohio State University, Dec. 1989.

[6] Nanette J. Boden, Danny Cohen, Robert E. Felderman, Alan E. Kulawik, Charles L. Seitz, Jakov N. Seizovic, and Wen-King Su. Myrinet – A Gigabit-per-Second Local-Area Network. *IEEE MICRO*, Feb. 1995.

[7] William J. Bolosky, Robert P. Fitzgerald, and John R. Douceur. Distributed schedule management in the tiger video fileserver. In *Sixteenth ACM Symposium on Operating System Principles*, volume 31, pages 212–223. ACM, December 1997.

[8] Solaris On-Line Documentation. http://www.docs.sun.com.

[9] Dawson R. Engler, M. Frans Kaashoek, and James O'Toole Jr. Exokernel: An operating system architecture for application-level resource management. In *Proceedings of the 15th Symposium on Operating System Principles*, December 1995.

[10] Edward W. Felten, Richard D. Alpert, Angelos Bilas, Matthias A. Blumrich, Douglas W. Clark, Stefanos Damianakis, Cezary Dubnicki, Liviu Iftode, and Kai Li. Early Experience with Message-Passing on the SHRIMP Multicomputer. *Proceedings of the 23rd International Symposium on Computer Architecture*, May 1996.

[11] D. Ferrari, A. Banerjea, and H. Zhang. Network support for multimedia. a discussion of the tenet approach. *TR-92-072*, 1992.

[12] Marc E. Fiuczynski, Brian N. Bershad, R.P. Martin, and D.E. Culler. SPINE - An Operating System for Intelligent Network Adapters. (TR-98-08-01), Aug. 1998.

[13] I_2O Special Interest Group. www.i2osig.org/architecture/techback98.html.

[14] Intel. *IQ80960Rx Evaluation Platform Board Manual*, March 1997.

[15] Michael B. Jones, Daniela Rosu, and Marcel-Catalan Rosu. Cpu reservations and time constraints: Efficient, predictable scheduling of independent activities. In *Sixteenth ACM Symposium on Operating System Principles*, volume 31, pages 198–211. ACM, December 1997.

[16] C.W. Mercer, S. Savage, and H. Tokuda. Processor capacity reservation for multimedia operating systems. In *IEEE International Conference on Multimedia Computing and Systems*. IEEE, May 1994.

[17] David Mosberger and Tai Jin. httperf – a tool for measuring web server performance. In *Proceedings of the 1998 Workshop on Internet Server Performance , held in conjunction with Sigmetrics 1998*, June 1998.

[18] I_2O Intel Page. http://www.developer.intel.com/iio.

[19] Scott Pakin, Mario Laura, and Andrew Chien. High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet. *Supercomputing*, Dec. 1995.

[20] Xingang Guo Pawan Goyal and Harrick M. Vin. A hierarchical cpu scheduler for multimedia operating systems. In *2nd Symposium on Operating Systems Design and Implementation*, pages 107–121. USENIX, 1996.

[21] Jennifer L. Rexford, Albert G. Greenberg, and Flavio G. Bonomi. Hardware-efficient fair queueing architectures for high-speed networks. In *INFOCOMM'96*, pages 638–646. IEEE, March 1996.

[22] Marcel-Cătălin Roşu, Karsten Schwan, and Richard Fujimoto. Supporting Parallel Applications on Clusters of Workstations: The Intelligent Network Interface Approach. *Proceedings of the 6th IEEE International Symposium on High Performance Distributed Computing*, Aug. 1997.

[23] Daniela Rosu, Karsten Schwan, and Sudhakar Yalamanchili. FARA - a framework for adaptive resource allocation in complex real-time systems. In *Proceedings of the 4th IEEE Real-Time Technology and Applications Symposium (RTAS)*, Denver, USA, June 1998.

[24] Daniela Rosu, Karsten Schwan, Sudhakar Yalamanchili, and Rakesh Jha. On Adaptive Resource Allocation for Complex Real-Time Applications. *18th IEEE Real-Time Systems Symposium*, Dec., 1997.

[25] Marcel-Catalin Rosu and Karsten Schwan. Sender coordination in the distributed virtual communication machine. In *Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing (HPDC '98)*. IEEE, 1998.

[26] Marcel-Catalin Rosu, Karsten Schwan, and Richard Fujimoto. Supporting Parallel Applications on Clusters of Workstations: The *Virtual Communication Machine*-based Architecture. *Cluster Computing*, 1:1–17, Jan. 1998.

[27] Karsten Schwan and Hongyi Zhou. Dynamic scheduling of hard real-time tasks and real-time threads. *IEEE Transactions on Software Engineering*, 18(8):736–748, August 1992.

[28] Thorsten von Eicken, Anindya Basu, Vineet Buch, and Werner Vogels. U-Net: A User-Level Network Interface for Parallel and Distributed Computing. *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, Dec. 1995.

[29] Thorsten von Eicken, David E. Culler, Seth Copen Goldstein, and Klaus Erik Schauser. Active messages: A mechanism for integrated communication and computation. *Proceedings of the 19th International Symposium on Computer Architecture*, May 1992.

[30] Jonathan Walpole, Rainer Koster, Shanwei Chen, Crispin Cowan, David Maier, Dylan McNamee, Calton Pu, David Steere, and Liujin Yu. "a player for adaptive mpeg video streaming over the internet". In *Proceedings 26th Applied Imagery Pattern Recognition Workshop AIPR-9*, Washington DC, October 1997.

[31] Richard West, Raj Krishnamurthy, William Norton, Karsten Schwan, Sudhakar Yalamanchili, Marcel Rosu, and Sarat Chandra. Quic: A quality of service network interface layer for communication in nows. In *Proceedings of the Heterogeneous Computing Workshop, in conjunction with IPPS/SPDP*, San Juan, Puerto Rico, April 1999.

[32] Richard West and Karsten Schwan. Dynamic window-constrained scheduling for multimedia applications. In *6th International Conference on Multimedia Computing and Systems, ICMCS'99*. IEEE, June 1999. Also available as a Technical Report: GIT-CC-98-18, Georgia Institute of Technology.

[33] Richard West, Karsten Schwan, and Christian Poellabauer. Scalable scheduling support for loss and delay constrained media streams. Technical Report GIT-CC-98-29, Georgia Institute of Technology, 1998.

[34] WindRiver Systems. *VxWorks Reference Manual*, 1 edition, February 1997.

[35] Hui Zhang and Srinivasav Keshav. Comparison of rate-based service disciplines. In *Proceeedings of ACM SIGCOMM*, pages 113–121. ACM, August 1991.