Leveraging Block Decisions and Aggregation in the ShareStreams QoS Architecture

Raj Krishnamurthy, Sudhakar Yalamanchili, Karsten Schwan, Richard West

Center for Experimental Research in Computer Systems
Georgia Institute of Technology

Atlanta, GA 30332

{rk}@cc.gatech.edu
Total Text: 12 pages of 20 pages (with figures, tables, refs)

October 11, 2002

Abstract

ShareStreams (*S*calable *H*ardware *Ar*chitectures for *Stream S*chedulers) is a canonical architecture for realizing a range of scheduling disciplines. This paper discusses the design choices and tradeoffs made in the development of a Endsystem/Host-based router realization of the ShareStreams architecture. We evaluate the impact of block decisions and aggregation on the ShareStreams architecture. Using processor resources for queuing and data movement, and FPGA hardware for accelerating stream selection and stream priority updates, ShareStreams can easily meet the wire-speeds of 10Gbps links. This allows provision of customized scheduling solutions and interoperability of scheduling disciplines. FPGA hardware uses a *single-cycle* Decision block to compare multiple stream attributes simultaneously for pairwise ordering and a Decision block arrangement in a recirculating network to conserve area and improve scalability. Our hardware implemented in the Xilinx Virtex family easily scales from 4 to 32 stream-slots on a single chip. A running FPGA prototype in a PCI card under systems software control can provide scheduling support for a mix of EDF, static-priority and fair-share streams based on user specifications and meet the temporal bounds and packet-time requirements of multi-gigabit links.

1 Introduction

A confluence of events is making QoS (Quality-of-Service) provisioning in clusters an exciting research area. First, (1) wire-speeds in clusters are steadily increasing with ubiquitous deployment of gigabit Ethernet NIs & switches, impending availability of Infiniband-2.5Gbps[2] and 10Gig Ethernet Alliance [1] hardware, and plans for Infiniband-10Gbps and 30Gbps hardware. Second, (2) workloads running on server clusters are increasingly a mix of best-effort web-traffic, real-time media streams, scientific and transaction processing workloads. Third, (3) Systems-on-a-chip (SoC) solutions [28] that combine a microprocessor datapath with a reprogrammable logic fabric [28] are now available, as are optimized datapaths for network

packet processing and transmission in network processors. Single-chip FPGAs[28] with customized logic capabilites for 10M gate designs and beyond are available at high clock-rates of 200MHz, supporting low reconfiguration overheads.

FCFS (First-Come-First-Serve) stream schedulers on end-system server machines or switches will easily allow bandwidth-hog streams to flow through, while other streams starve. Flexible scheduling disciplines form the heart of effective QoS provisioning in clusters, where real-time media streams, best-effort web traffic and general-purpose workload traffic can effectively be served together. Availability of tightly-coupled processor and reconfigurable logic solutions means that customized scheduling solutions can be provided, with the flexibility of software at hardware speeds. This ensures matching the needs of the constantly changing landscape of network services and protocols. Our previous experiences with host-based and embedded software schedulers [27, 12], has shown that meeting packet-time requirements of multi-gigabit links is difficult with software-only realizations of scheduling disciplines. Scheduling disciplines must be able to make a decision within a *packet-time* ($\frac{packet-length(in\ bits)}{line-speed(bps)}$), to maintain high link utilization.

These trends call for a architectural framework that allows us to reason about providing packet scheduling solutions, balancing performance & constraints and making tradeoffs required in their physical realization. The ShareStreams architecture [15, 13] is a unified canonical architecture for supporting a range of scheduling disciplines from priority-class & fair-queuing to the more recent window-constrained scheduling disciplines [26]. Figure 1 shows a relationship between QoS bounds, scale (number of streams or granularity or aggregation degree) and scheduling rate. For serving a large number of streams, with pre-determined QoS bounds (bandwidth, delay and delay-jitter), a higher scheduling rate might be needed to select a stream and adjust the priority of streams. Similarly, scheduling and serving MPEG frames (with larger granularity and larger packet-times than 1500-byte or 64-byte Ethernet frames) may not require a high scheduling rate. An architectural solution may be able to provide QoS bounds for a given number of streams (N) & granularity (packet-size) at a certain scheduling rate. Figure 1(b) shows, if at all, the required scheduling rate, can be realized in silicon or reconfigurable logic, given the implementation complexity of a given scheduling discipline. By similar argument, if only a common-case or lower than required for worst-case scheduling rate can be realized, what will be the degradation in QoS?. Furthermore, will this be acceptable to applications?, if more streams or smaller packets need to be serviced while the scheduling discipline is in operation.

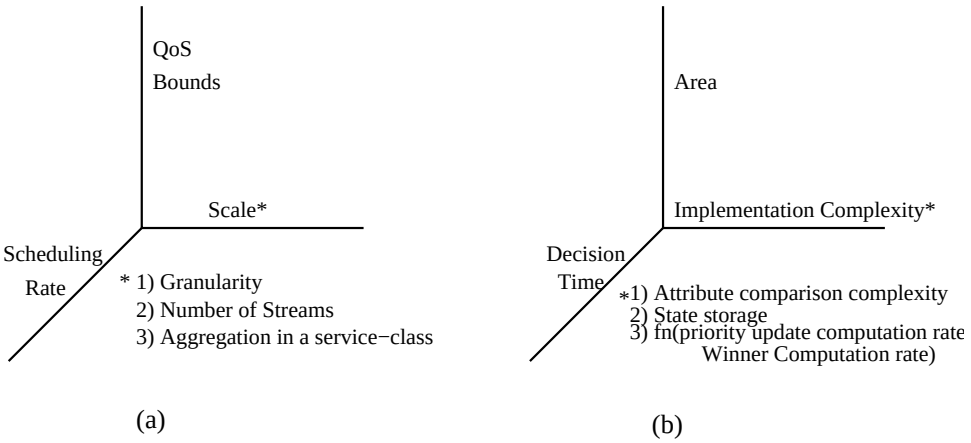

Figure 1: (a) ShareStreams Architectural Solutions Framework (b) Implementation Complexity of Packet Schedulers

The SHAreStreamS (*S*calable *H*ardware *Ar*chitectures for *Stream S*cheduling)[16, 15, 13] shown in Figure 3 combines a commercial microprocessor datapath or network processor with a reconfigurable logic fabric. The complexity of stream selection and priority update computations poses a challenging implementation problem for scheduling a large number of streams over multi-gigabit links. For example, the Ethernet frame time on a 10 Gigabit link ranges from approximately 0.05 microseconds (64 byte) to 1.2 microseconds (1500 byte). This can be substantially lower for ATM cells or SONET frames that need to be scheduled at wire speeds. Packet level QoS scheduling at these link speeds poses significant implementation challenges. Our architecture stores per-stream state and attribute adjustment logic in Register base blocks and orders streams pair-wise using multi-attribute-compare-capable Decision blocks in a recirculating shuffle network to conserve area. Scheduling logic does possess significant amount of parallelism for which we propose a customized FPGA solution. Such solutions are viable as FPGA technology pushes 10 M gate designs with clock rates of up to 200MHz with relatively low reconfiguration overheads. By carefully crafting suitable implementations for compute-intensive scheduler components for implementation within the FPGA, we find tractable implementations for the fine grained, real-time packet scheduling problem. Motivation for choice of architectural components is discussed in Section 4.

Contributions The focus of this paper is discussing the design choices and tradeoffs in development of a unified canonical architecture for a range of scheduling disciplines, evaluating the ShareStreams Endsystem realization, and the impact of two specific tradeoffs on performance. One tradeoff relates to use of the sorted list of streams available in the ShareStreams architecture at the end a *decision-cycle*, called a *block*. This can improve scheduler throughput by a factor of *block* size. The alternative is to route only the stream with the highest priority and reduce physical interconnect requirements. The second tradeoff relates to achieving scale by aggregating streams using processor resources, and thereby saving FPGA resources. Since ShareStreams is a unified canonical architecture for a range of scheduling disciplines, the design choices and tradeoffs discussed in this paper will be useful for researchers and architects building and evaluating a range of QoS architectures. The paper provides comparisons with contemporary queuing and scheduling architectures. In [15, 16], we presented the ShareStreams line-card realization and focused on the hardware architecture. This paper focuses on system-level design issues in the tandem operation of ShareStreams hardware and software.

Outline of Paper. The paper begins by describing the ShareStreams architectural framework for making tradeoffs in realization and achieving QoS bounds, in Section 2. Section 3 discusses related queuing and scheduling architectures. The ShareStreams endsystem architecture, along with hardware and software architecture design choices & tradeoffs is discussed in Section 4. Evaluation of the impact of *block* decisions and aggregation is described in Section 5 along with the demonstration of the ShareStreams Endsystem architecture. Section 6 concludes the paper with a description of future work.

2 The ShareStreams Architectural Framework

This Section describes the complexity and issues in the design of architectures for packet schedulers and introduces the ShareStreams unified canonical architecture.

Packet Schedulers: Properties and Complexity The fundamental idea in packet scheduling is to pick a stream from a given set of streams and schedule the head-packet from the eligible stream for transmission. The scheduling discipline must make this decision based on stream service constraints (reduced to *service-tags*), expressed as descriptors/attributes (which could be integer-valued weights by which bandwidth of the output link is to be divided or deadlines at which packets in each stream may need service) so that the service

requirements of each stream (bandwidth, delay or jitter) are satisfied to the best extent possible. The stream attributes of relevance to a certain scheduling discipline by which streams are ordered may be multi-valued (deadlines, loss-ratios) or single-valued (stream weights) and may be abstracted for convenience as stream priorities. A static priority scheduling discipline (which minimizes the weighted mean delay) for non-time-constrained traffic picks a stream based on a static time-invariant priority. A dynamic priority scheduling discipline on the other hand, will bias or alter the priority of streams every scheduling decision cycle so that streams waiting for service may also be picked (albeit eventually) over the stream recently serviced. Table 1 compares priority-class scheduling disciplines (used in DiffServ[7]), fair queuing scheduling disciplines and window-constrained scheduling disciplines along different dimensions. In fair-queuing schedulers [6, 29], a

Characteristic	**Priority-class**	**Fair-queuing** WFQ, SFQ	**Window-constrained** (m,k)-firm[8], DWCS
Priority Grain	Stream-level dynamic Packet-level fixed	Stream-level dynamic Packet-level fixed	Stream-level dynamic Packet-level *dynamic*
Input Queue	Priority Queue	Priority Queue	Simple circular queue
Service-tag Computation	concurrent across streams	per-stream serialized	winner in previous decision cycle
Concurrency	Multiple decisions can be pipelined	Multiple decisions are pipelined	Successive decisions are serialized

Table 1: Comparing Scheduling Disciplines

service-tag (start-time or finish-time) is assigned to every incoming packet and packets with the least service tag are served first. In DWCS[26] (Dynamic Window-constrained Scheduling), deadlines and loss-ratios are used for servicing packets based on rules shown in Table 2. Service attributes of packets are updated every scheduling decision cycle and multiple stream service attributes are used for every *decision*. This allows window-constrained scheduling disciplines to provide support for EDF (Earliest-Deadline First), Fair-share and static-priority streams.

Key Insight for a Unified Architecture A key insight here is that for priority-class and fair-queuing scheduling disciplines, the packet priority does not change after each packet is queued. In window-constrained scheduling the packet priority can change every scheduling decision cycle. So a unified architecture for the spectrum of scheduling disciplines can be developed by providing support to update packet priorities every decision cycle to allow mapping of window-constrained scheduling disciplines. For supporting fair-queuing and priority-class scheduling disciplines, the packet priority update cycle is simply bypassed.

DWCS Background Every stream requiring service is assigned two service attributes - a Deadline and a window-constraint or loss-tolerance (ratio) (W_i). A request period (T_i) is the interval between deadlines of two successive packets in the same stream (S_i). The end of a request period (T_i) is the deadline by which the packet requiring service must be scheduled for transmission. The window-constraint (W_i) or loss-tolerance ($\frac{x_i}{y_i}$) is the number of packets x_i (loss-numerator) that can be late/lost over a window y_i (loss-denominator) packet arrivals in the same stream S_i. All packets in the same stream have the same loss-tolerance or window-constraint (W_i) but a different deadline (separated by the request period). In order for a *winner* or eligible stream to be picked, the streams must be ordered pairwise based on the rules presented in Table 2. Streams that miss deadlines (*losers*) have their priorities adjusted (in effect to raise their priorities), different from the *winner* stream (which has priority effectively lowered).

Pairwise Ordering for Streams
Earliest-Deadline First
Equal Deadlines, order lowest window-constraint first
Equal deadlines and zero window-constraints, order highest window-denominator first
Equal deadlines and equal non-zero window-constraints, order lowest window-numerator first
All other cases: first-come-first-serve

Table 2: Example Scheduler Decision Rules

ShareStreams Architectural Framework & Canonical Architecture

The implementation complexity of a dynamic-priority scheduling discipline is shown in Figure 1 (b) and dependent on the following factors.

State Storage. The service attributes that the scheduling discipline is dependent on must be updated and stored as streams are scheduled, along with scheduler specific counters and flags e.g. packet discard flags and scheduling discipline performance counters.

Attribute Comparison Complexity. Certain scheduling disciplines like EDF (Earliest-Deadline-First) and WFQ[6](Weighted Fair-Queuing) use only one attribute for comparison namely, deadlines or stream weights. Flexible scheduling disciplines like DWCS use a combination of multiple attributes like deadlines, loss-ratios and arrival times for comparison and pairwise stream ordering. This determines the decision rate.

Winner Selection Rate & Priority Update Rate. Ultimately, all streams have to be ordered and the highest "priority" stream must be picked, which determines the winner selection rate. State storage and update operations on service attributes determine the priority update rate. In DWCS for example, the stream service attributes are updated every scheduling decision cycle.

The ShareStreams architecture stores stream state in Register Base blocks or Stream-slots, stream service attributes are compared pair-wise using Decision blocks and streams are sorted in $log_2 N$ cycles using a recirculating shuffle-exchange network. The ShareStreams architecture is a unified canonical architecture to realize priority-class, fair-queuing and window-constrained scheduling disciplines as shown in Table 1. ShareStreams is able to achieve this by providing support for state storage, decisions and stream ordering, which are primitive operations required across all scheduling disciplines. Figure 1 shows the ShareStreams architectural framework which helps in choice of architectural components required for a certain application. *QoS Bounds* and *Scale* needs of a certain application, might stipulate a certain scheduling rate. The ShareStreams architectural components - processor, FPGA and different blocks in the FPGA can be chosen to meet this *Scheduling Rate*. The implementation complexity of the scheduling discipline with chosen architectural components will determine the achievable scheduling rate. The ShareStreams framework allows trade-offs to be made to achieve QoS and scale goals. This paper focuses on two such tradeoffs - block decisions and aggregation.

The next Section describes related work by discussing queuing and scheduling architectures used in software-based routers and high-end switches & routers used in the Internet's core.

3 Related Work : Queuing and Scheduling Architectures

Queuing and Scheduling architectures have evolved separately for host-based routers and backbone switches & routers. A common performance requirement is that queuing, switching/routing and output-link scheduling be completed at wire-speeds to maintain high link-utilization and scale.

Software-based Routers

Software-based routers provide low-cost solutions for routing within the Internet's edge with modest bandwidth requirements. Here meeting packet-times for multi-Gbps and 10Gbps links is usually not possible as described in Section 4.1, and these systems are mostly targeted towards multimedia applications, firewalls, network intrusion and dataplane applications with modest bandwidth requirements. A number of recent projects have developed architectures, analyzed performance bottlenecks and provided flexible extensions for software-based routers [5, 19, 11, 22] on end-systems. The focus has been on extensibility and performance for data forwarding IP-protocol applications, rather than Ethernet frames for multi-Gbps links. While QoS capabilities with Deficit Round Robin[5] and H-FSC[23] have been studied in [5, 19, 11], [22] focuses on software architectures for high-rate forwarding in Network processors without considering impact of complex scheduling disciplines.

Hardware Priority Queuing Architectures

A number of hardware structures have been proposed to implement traditional priority queues. In traditional priority queuing systems, a packet arrives and is given a priority or a service tag (start/finish number) based on the state of each of the backlogged queues or a round number / virtual time-stamp representing the run-time progression of scheduling. [10], [18], [3], all propose interesting priority queuing structures. None of these architectures can be used to provide a unified canonical architecture for priority-class, fair-queuing *and* window-constrained schedulers. First, a heap, a systolic queue or a shift-register chain implementation will require replication of the ShareStreams Decision block in every element. The ShareStreams recirculating shuffle conserves area by using only the lowermost-level of a tree. Note that ShareStreams Decision blocks require multiple service attributes to be compared simultaneously and are not simple comparators. Second, the priorities of streams that miss deadlines and the winning stream are updated every *decision-cycle*. This will require resorting the heap, systolic queue and shift-register chain (formed from each arriving packet) every decision-cycle *and* on packet-arrival. A simple binary tree simply wastes area, and requires $log_2(N)$ *levels* of the tree. Instead, a recirculating shuffle, used in our canonical architecture conserves area, and *scales* better by using only ($\frac{N}{2}$) decision blocks in a single-stage recirculating shuffle. A single-stage recirculating shuffle network can be used with priority-class based scheduling disciplines and also with fair-queuing scheduling disciplines as N packets with service-tags can be ordered in $log_2 N$ cycles, using simple comparators to compare stream weights. An extra priority update cycle is not needed.

4 The ShareStreams Hardware & Software Architecture

Scheduling disciplines are a central component of QoS architectures and strive to provide bandwidth, delay and jitter guarantees for streams. Development of architectural solutions for wire-speed scheduling disciplines requires a prudent choice of architectures. This allows each component of the scheduling discipline to be run with high performance given a set of implementation constraints. We first motivate the choice of architectural components in the ShareStreams architecture and describe the hardware & software architecture. Two practical realizations of the architecture are also described along with a prototype used for performance evaluation.

4.1 Performance and Limits of Processor-resident Packet Schedulers

A natural first choice to run a packet scheduling discipline is on a processor. Given that the ShareStreams architecture must run a range of scheduling disciplines and also meet wire-speeds, choice of a processor must allow priority-class based disciplines, service-tag based fair-queuing disciplines and window-constrained scheduling disciplines to be run at wire-speeds. Priority-class based schedulers have lower computational complexity, while service-tag based schedulers need calculation of a virtual finish-time which can be complex. Window-based schedulers require priority-updates every decision cycle and have a high computational complexity, but can provide extremely sophisticated scheduling for a mix of traffic streams. Detailed performance studies on Sun Ultrasparc 300MHz processors, completed in [27] show that the scheduler latency can be as high as $\approx 50\mu$s for window-constrained scheduling disciplines. Similarly we show in [12] that for a 66MHz i960RD processor, the scheduler latency is around $\approx 67\mu$s on a more lighterweight Operating System kernel. This is clearly unsuitable for 10Gbps wire-speed scheduling disciplines. Results in [5] on a 233MHz Pentium show packet processing overhead using the Deficit Round Robin scheduling discipline of $\approx 35\mu$s run in the NetBSD kernel. Measurements from [23] show packet processing overhead of the order of $\approx 7 - 10\mu$s on a 200MHz Pentium using the Hierarchical Fair-service curve scheduling discipline. This is suitable for meeting packet-times of a 1 Gbps link for 1500-byte frames (12μs), but not 64-byte ($500ns$) frames.

4.2 Making Architectural Choices for Wire-Speed Scheduling Discipline Operation

Processor-resident packet schedulers can barely meet the packet-time requirements of even 1 Gbps links as described above and are more suited to scheduling in end-systems and host-based routers. The ShareStreams Endsystem realization for host-based routers queues packets on the processor-memory subsystem. The data transfer engines on Network processors and DMA engines on the Network Interface are leveraged for high throughput packet transfers. An FPGA array is used to accelerate pairwise ordering decisions between stream priorities and winner stream selection to meet the packet-time requirements of links. This is a prudent architectural choice as a number of systems-on-a-chip solutions are being announced with a combination of processor and FPGA resources in a tightly-coupled fashion. Also FPGA co-processors can be placed on the I/O bus and peer-peer transfers can be completed with high-rates on modern backplane buses like PCI from a Network processor. Such architectures are suitable for low-end switches and routers. Meeting packet-times for 10Gbps links is critical for backbone switches and routers and the ShareStreams line-card realization shown in Figure 2, uses dual-ported memory between the switch fabric and the FPGA scheduler to receive packet arrival-times and provides Stream IDs to the network transceiver. The line-card realization is critical for operation in a network backbone where thousands of streams are switched and routed by network hardware.

Motivating Decisions and Stream Selection on FPGAs A scheduling discipline will pick a stream from a set of streams based on service attributes independently expressed by each stream, to meet bandwidth, delay and jitter guarantees. There is high degree of parallelism at the level of a stream and at the packet-level. A hardware implementation must explicitly capture this stream-level and packet-level parallelism. Picking a stream from a set of streams requires pairwise ordering of streams and hardware concurrency can provide separate datapaths realized in logic to compare service attributes from two different streams. Note that multiple service attributes can be compared concurrently in hardware allowing single cycle comparison between stream priorities. Also state storage for individual streams can be provided in registers realized in FPGA CLB (Configurable Logic Block) flip-flops, with the added advantage that updates to register values can be

completed concurrently. Also interconnect density on a FPGA allows different components to be connected with wide-buses. FPGAs like Xilinx Virtex II Pro have multiple Power-PC cores[9] immersed in a CLB array. FPGAs provide a convenient hardware implementation substrate with required price/performance, clock-rates upto 200MHz, and 10 million gate densities without ASIC engineering overheads. The flexibility of software at hardware speeds, available with FPGAs, allows scheduling disciplines to evolve with the constantly changing landscape of network services and protocols.

ShareStreams Linecard Realization A line-card realization of the ShareStreams architecture is shown in Figure 2. Dual-ported SRAM allows packets arriving from the switch-fabric to be placed in per-stream SRAM queues. Their arrival times can be read by the SRAM interface concurrently. Winner Stream IDs are written into the SRAM partition by the SRAM interface, which are provided by the Scheduler control unit.

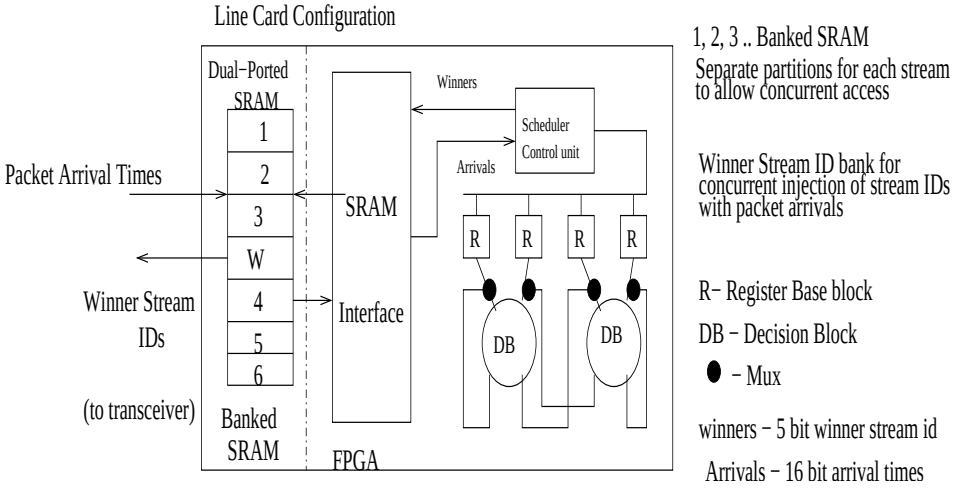

Figure 2: ShareStreams Switch Linecard Realization

ShareStreams Endsystem/Host-Router Realization

The ShareStreams Endsystem realization is targeted for endsystems and host-based routers where meeting individual packet-times at the level of an Ethernet frame may not be essential. Here forwarding throughput for IP frames and jumbo IP frames is important to provide multimedia streaming rates of tens of frames every second and other applications like telepresence, surveillance, firewalls, intrusion-detection [17, 24]. To achieve this, ShareStreams uses the processor in a endsystem for packet queuing and data movement and the FPGA as a co-processor to complete decisions and stream selection.

The Stream Processor and Systems Software The ShareStreams architecture maintains per-stream queues usually created on a stream processor (see Figure 3) by a Queue Manager (QM). A Stream processor is the host processor or a Network processor (like the IXP 1200) on the I/O bus. ShareStreams' per-stream queues are circular buffers with separate read and write pointers for concurrent access, without any synchronization needs. This allows a producer to populate the per-stream queues, while the Transmission Engine (TE) may concurrently transfer scheduled frames to the network. As streams arrive, their service attributes or constraints are transferred to the FPGA PCI card, where the ShareStreams scheduler hardware and streaming unit are resident. The data is deposited in banked SRAM for concurrent access between the Stream processor and ShareStreams scheduler hardware. The Stream processor communicates 16-bit arrival-time offsets to the Scheduler hardware unit (not the packets themselves) and reads/receives 5-bit Stream IDs of scheduled

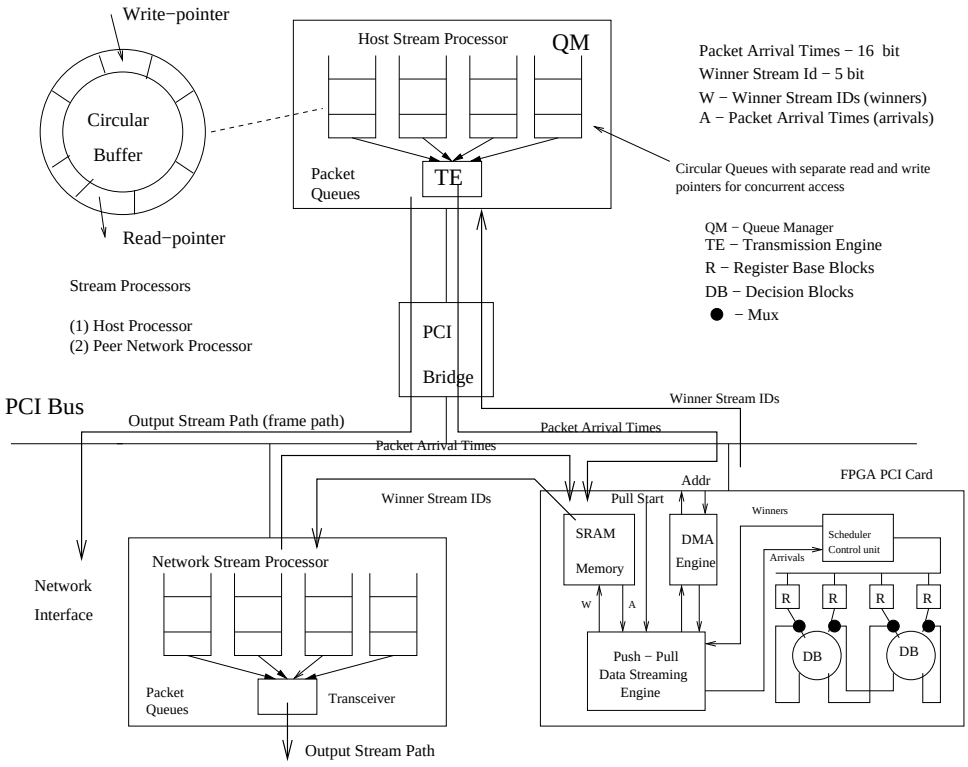

Figure 3: ShareStreams Endsystem Realization

streams. Per-stream buffering is provided in SRAM banks on the card and block RAMs on the chip itself. Transmission Engine(TE) threads are responsible for enabling transfer of packets in scheduled streams to the network (set DMA registers on NI to enable DMA pulls). ShareStreams attempts to maintain concurrency between packet queuing, scheduling and transmission by separating functions between the Stream processor and the Scheduler hardware unit. The Scheduler hardware and Streaming unit are described next.

ShareStreams Hardware and Streaming Unit interacts with Systems Software The Scheduler hardware and Streaming unit are resident on the FPGA PCI card. The Scheduler hardware consists of a Control unit, per-stream state storage Register blocks (or Stream-slots) and Decision blocks arranged in a recirculating shuffle-exchange network (see Figure 4). Register Base blocks, also called Stream-slots to capture the effect of aggregating many streams to a Register Base block, if only aggregate QoS is required for a set of streams rather than on a per-stream basis. The Streaming unit keeps per-stream queues on the FPGA PCI card *full* using a combination of *push* and *pull* transfers. For small transfers, the Stream processor can push arrival-times to the FPGA PCI card. For bulk-transfers, the Stream processor will set the DMA engine registers and assert the *pull-start* line so that bank ownership can be arbitrated between the Stream processor and the Scheduler hardware unit. ShareStreams provides support for efficient exchange of arrival-times and Stream IDs between the Stream processor and the Scheduler Hardware unit. The hardware architecture of the Scheduler hardware unit is described in the next Section.

Summary of Design Issues and Choices An efficient packet store with queuing is essential for host-router designs along with efficient data transfer from the processor-memory subsystem to the network. Concurrency between queuing, scheduling and transmission is important to meet packet-time requirements of outgoing links by using synchronization-free constructs, efficient transfers for exchanging data with the

co-processor and transferring data between the packet store and outgoing network links. For a scheduling architecture, pipelining multiple stream selection decisions is crucial to maintain high throughput. Also pairwise ordering of streams can be done efficiently by concurrent evaluation of ordering rules.

4.3 ShareStreams Unified Canonical Architecture

This Section describes the FPGA hardware architecture of the ShareStreams Scheduler on Xilinx Virtex FPGAs. The ShareStreams scheduler must be able to allow mapping of priority-class, fair-queuing and window-constrained scheduling disciplines. As described in Section 2, window-constrained scheduling disciplines update their stream priorities every *decision-cycle*. Also multiple stream attributes from streams are compared simultaneously for pairwise ordering, which require *decision-blocks* rather than simple comparators. Fair-queuing disciplines do not update packet priorities every decision cycle and require simple comparators to compare weights.

Motivating Choice of State Storage and Network Hardware heaps, systolic queues and shift-register chains are commonly used to realize fair-queuing and priority-class disciplines. Using hardware heaps, systolic queues or shift-register chains for window-constrained scheduling leads to needing a re-sort of the heap or systolic queue every decision-cycle. Also relatively complex decision-blocks need replication in each element of the heap, systolic queue or shift-register chain. Instead, the ShareStreams architecture arranges the Decision blocks in a recirculating shuffle-exchange network. Streams are ordered pairwise yielding a sorted list of streams every decision cycle. Each decision cycle takes $log_2(N)$ hardware cycles for N streams. Arranging Decision blocks in a tree *wastes* area as successive levels of a Decision block tree cannot be pipelined with window-constrained schedulers. At the end of a decision cycle, the winner must be circulated to the stream state store, so that stream priorities can be updated. This requirement disallows the binary Decision block tree to be pipelined. Arrangement of decision blocks in a recirculating shuffle-exchange network, requires only ($\frac{N}{2}$) decision blocks (only one level of the equivalent Decision block tree). This arrangement is also beneficial to priority-class or fair-queuing scheduling disciplines as packet service-tags across all N streams can be ordered in $log_2(N)$ cycles in *block* fashion, without a priority update cycle.

Mapping a Window-constrained Scheduling Discipline To demonstrate the versatility of our architecture, we implement a dynamic priority packet scheduling algorithm - DWCS[26], which requires a priority update or service attribute update every decision cycle. This allows the packet scheduling algorithm to service static-priority, fair-share and EDF streams or a combination thereof, using a single architecture realization[13]. Per-stream service attributes are stored in Register Base blocks, Decision Blocks allow pairwise comparison of two streams, using multiple stream service attributes. Recirculating the stream service attributes allows pairwise ordering of all streams in $log_2(N)$ cycles for N stream Register Base blocks, using a single-stage recirculating shuffle-exchange network. Register Base blocks are also called *Stream-slots* to capture the effect of choosing to aggregate many streams to a Register Base block, if only aggregate QoS is desired for a set of streams rather than on a per-stream basis. A sorted list of streams is obtained after $log_2(N)$ cycles and the winner ID is circulated to every Register Base block so that per-stream updates can be applied based on whether a stream is a winner or a loser or whether a stream has missed or met its deadline. The network requires N Register Base blocks, ($\frac{N}{2}$) Decision blocks and $log_2(N)$ cycles of the recirculating shuffle-exchange network for determination of a winner stream. Details of the hardware implementation of ShareStreams with DWCS is described in [13, 15]. Figure 4 shows the hardware implementation of DWCS mapped to ShareStreams. Register Blocks supply stream service attributes (16-bit packet deadlines, 8-bit loss numerator, 8-bit loss denominator, 16-bit arrival times and 5-bit Register IDs)

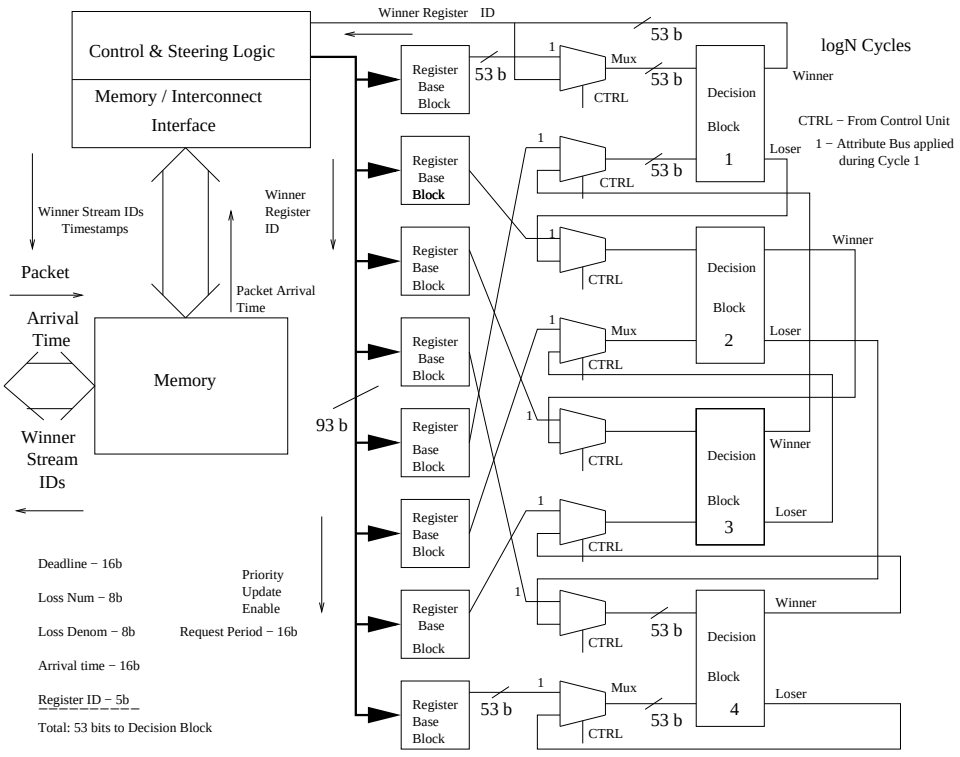

Figure 4: ShareStreams Hardware Architecture: Recirculating Shuffle (All Field Lengths in Bits)

to Decision blocks which compare stream attributes between two streams in a *single* cycle. Decision blocks are shown in Figure 5, which implement the rules in Table 2 by evaluating all possibilities concurrently and choosing the output of the valid rule based on the input data. Muxes control data applied to each Decision block every cycle, and are set by the Control and Steering Logic unit. The Control and Steering logic unit loads the Register Base blocks and interfaces with the memory unit for exchanging packet arrival-times and Stream IDs. The Control and Steering logic unit begins in a LOAD state and then alternates between SCHEDULE and PRIORITY_UPDATE states as shown in Figure 6.

Mapping Priority-class and Fair-queuing Schedulers Fair-queuing schedulers use per-packet service-tags to order streams. The per-packet service-tags do not change once they are computed. The ShareStreams architecture simply needs to use the LOAD and SCHEDULE states, as per-packet service-tags will not change. The architecture can order N service-tags in $log_2 N$ cycles.

Max-finding(Winner-only Routing) and Block Decisions The output of a Decision block are winner and loser stream attributes that are routed to another Decision block in a successive cycle. Routing both *winners* and *losers* yields a sorted list of streams and this is termed a *block*. Conceivably, this block may be used to schedule packets in future packet-times, if bandwidth, delay and jitter guarantees can still be maintained. If this is possible, then the throughput of the scheduler increases by a factor equal to the *block*-size. By routing only winner streams out of a Decision block in Figure 4, at the end of $log_2 N$ decision cycles, a single winner stream is available for transmission. This eases the physical interconnect requirements without the scheduler throughput improvement that can be provided by a block decision. Section 5 analyzes trade-offs concerning the use of blocks and the architecture in *max-finding* configuration with *winner-only* routing.

Stream Aggregation If aggregate QoS is required over a set of streams without any per-stream QoS, then

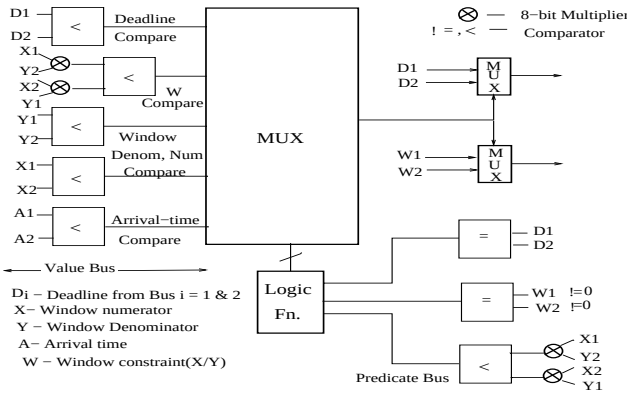

Figure 5: Decision Block: Concurrent Single-Cycle Evaluation

many streams (called streamlets, if aggregated) can be bound to a single Register Base block or *Stream-slot*. This is a powerful strategy to achieve scale by trading lower QoS bounds for higher stream count, or processor memory footprint size for lower FPGA state storage. Stream aggregation is evaluated in Section 5.

ShareStreams Hardware Prototype The hardware is run on a Celoxica Virtex I/1000 PCI card which can clock a design upto 100MHz and is equipped with a 32bit/33MHz PCI controller. The card is equipped with a 8M SRAM accessible from both a host/PCI peer and the Virtex FPGA with suitable arbitration (between the FPGA and host-PCI peer) provided by the firmware. Details of the card are found in [4].

5 Performance Evaluation

This Section compares Block scheduling and Max-finding configurations of the ShareStreams architecture, and describes performance and usability tradeoffs. The performance of the ShareStreams architecture is demonstrated in an Endsystem configuration, and described with tradeoffs between aggregation of streamlets into a stream-slot and per-streamlet QoS state storage. We use the ShareStreams-DWCS architectural realization and run the architecture in suitable modes for EDF, static-priority and fair-share operation[13].

5.1 Comparing Max-finding and Block Decision Architectural Configurations

Figure 7 shows the area and clock-rate of the Base architecture (BA, provides sorted-list) and the max-finding architecture (WR, winner-only routing), as explained in Section 4. For the purposes of Figure 7, we included only the Register Base blocks, Decision blocks, Control-steering Logic block and the recirculating shuffle-exchange network. The memory and interconnect interface was not included. Figure 7 shows the area and clock-rate of two design variants with the Xilinx Virtex I architecture (the Celoxica PCI card used in our experiments has a Xilinx Virtex I chip). A Virtex 1000 part has an equivalent of 1 million system gates with 64 x 96 Virtex I CLBs (2 Virtex I slices = 1 Virtex I CLB). A slice includes LUTs and flip-flops and is the basic logic element. The area of the Control-steering logic block was 22 slices, the Decision block was 190 slices and the Register Base block was 150 slices. The area of the shuffle-network wires and pass-through CLBs is dependent on the stream-slot count of a given design. Our architecture grows linearly, in terms of area, and our physically placed & routed designs also exhibit the same rate

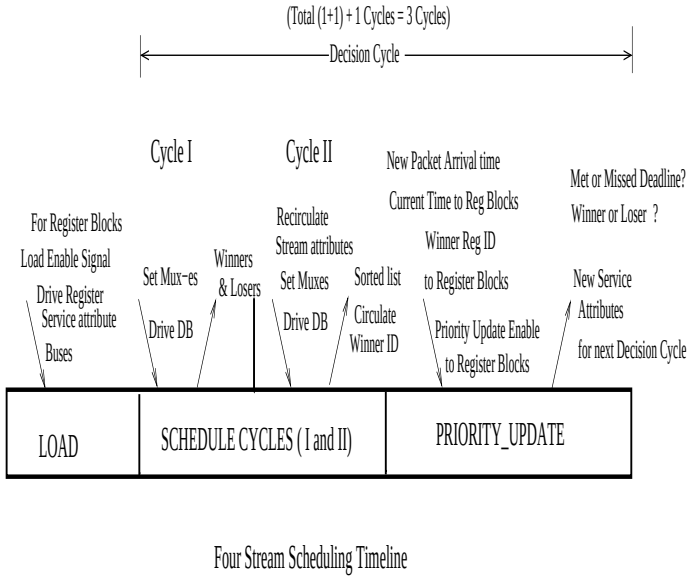

Figure 6: Sharestreams Scheduler Timeline

of growth for BA and WR configurations shown in Figure 7. Decision-time grows logarithmically in the ShareStreams hardware architecture, with 2, 3, 4, 5 cycles required to sort 4, 8, 16 and 32 stream-slots using the recirculating shuffle-exchange network. The WR architecture shows lesser clock-rate variation from 4 to 32 stream-slots, than the BA architecture. This can be attributed to easing the physical routing requirements by routing only winners, leading to a more compact logic spread. *Our Virtex I implementation can easily meet the packet-time requirements of all frame sizes (64-byte and 1500-byte) on gigabit links, and 1500-byte frames on 10Gbps links.* An interesting outcome of this evaluation is that the BA architecture, which allows *block scheduling*, shows only 10% degradation in clock-rate from its *winner-only* routed counterpart, for 32 streams. The BA architecture maintains almost the same area with it's WR counterpart for all stream-slot sizes.

Summary So unless specific packet-time requirements are to be met, for example 8 and 16 stream-slot sizes where the clock-rate degradation is close to 20% from WR to BA, the Base architecture (BA) can almost always be used with additional benefits as demonstrated below.

Block Decisions for Deadline-constrained Real-time Streams

Block scheduling allows stream priorities to be ordered every scheduler decision cycle across all streams using the Base architecture (BA), and generates a sorted list of streams using the sorting shuffle-exchange network. The winner-only (WR) routed variant, routes only winners and generates only the *max* priority stream-slot every decision-cycle. Block scheduling is inherently advantageous as *all* streams can be ordered every scheduler decision cycle. For fair-share streams requiring fair bandwidth allocation, transmitting the block with ordered list of streams on the single outgoing link can skew bandwidth allocations considerably. For real-time streams with deadline-constraints, block scheduling can be extremely useful as all streams can be ordered in $log_2 N$ cycles of the recirculating shuffle. The output *block* can be scheduled on the outgoing link in a *single* transaction, with missed deadlines being registered in performance counters for each stream/stream-slot. This is possible because deadlines of queued packets do not change during scheduling discipline operation. By ordering all stream deadlines in a single decision cycle, the throughput of the scheduler increases by a factor of the block size (can be equal to the number of stream-slots). Table

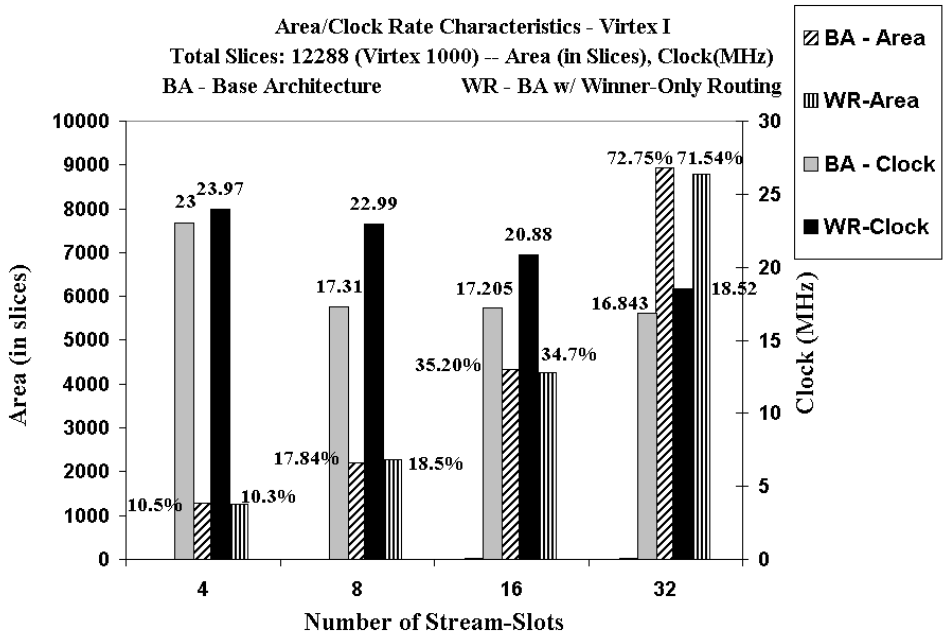

Figure 7: Area-Clock Rate Characteristics of Architecture (Virtex I)

3 compares the *winner-only* routing architectural configuration, which allows max-finding with the *block* scheduling architectural configuration. For the results in Table 3, we scheduled four streams, with one stream mapped to each stream-slot. In the max-finding configuration, we assigned each of the four streams successive deadlines that are one time unit apart. Each stream was requested every decision-cycle ($T_i = 1$). The ShareStreams-DWCS scheduler was set in EDF (Earliest-Deadline-First) mode. This leads to a number of missed deadlines for each stream, as each stream is requested every decision cycle. Only one stream can be picked in the max-finding configuration every decision cycle, while others with conflicting deadlines will increment their missed deadline counters by one. Table 3 also shows the block scheduling configuration in two modes - *max-first* and *min-first*. In the *max-first* mode, the stream with the highest priority is circulated during the PRIORITY_UPDATE cycle (as shown in Figure 4). In the *min-first* mode, the stream with the lowest priority ie. at the end of the *block* is circulated during the PRIORITY_UPDATE cycle. Table 3 shows that *max-first* mode allows all deadlines to be met, as streams with conflicting deadlines can be scheduled together in a block, along with streams requiring service in future packet-times. Block scheduling improves the throughput of the scheduler, only 16000 decision cycles are needed to schedule 64000 frames. The decision cycles shown in Figure 3 for the *block* case reflect those cycles, where the stream was a winner. The ShareStreams hardware architecture on the other hand, in max-finding configuration needs 64000 decision cycles to schedule 64000 frames.

Fair Bandwidth Allocation for Streams We use the target architecture shown in Figure 3 for the case of a End-system or host-based router. The host is a Pentium III 500MHz with Redhat Linux version 2.4. We run the ShareStreams hardware scheduler on a Celoxica FPGA PCI card[4] on a attached PCI bus. The host processor functions as the Stream processor for this target architecture.

Efficient Data Transfers are Critical for Exchanging Arrival-times and Stream IDs The Queue Manager (QM) on the Stream processor provides per-stream queues and stores service attributes in descriptor fields for each stream. Stream service constraints are communicated to the FPGA on the PCI bus by depositing in

Stream-Slot	Max-finding (winner-only)		Block (sorted-list)		
	Missed Deadlines	Decision cycles	Max-first Missed Deadlines	Min-first Missed Deadlines	Decision Cycles (winner)
Stream 1	63986	16000	0	27839	4000
Stream 2	63987	16000	0	27214	4000
Stream 3	63988	16000	0	22621	4000
Stream 4	63989	16000	0	29311	4000
Total	255950	64000	0	106985	16000

Table 3: Comparing Block Decisions and Max-finding

a special SRAM partition, we set service constraints to achieve a 1:1:2:4 bandwidth ratio allocation. A key requirement is that packet arrival-times from stream queues be provided to the FPGA and scheduled Stream IDs read by Queue Manager (QM) threads. Buffering is easily provided on the FPGA card SRAM banks and on-chip block RAM for packet arrival times and Streams IDs. Packet queues on the FPGA SRAM must be kept full as packets arrive in Stream processor queues by efficient transfers. Packet arrival-times are batched and transferred to the FPGA PCI card to take advantage of the burst PCI bandwidth with *push* transfers for small transfers and *pull* transfers from the PCI card for bulk transfers using PCI card DMA engines. Similarly, scheduled Stream IDs are read from the SRAM banks by the Stream processor using push and pull transfers. We use the TPIL[14] communications library developed as part of the Active System Area Networks project at Georgia Tech [14]. Providing concurrent accesses to the SRAM bank for the Stream processor and FPGA are crucial to providing high-performance. Note that the processor and FPGA exchange 16-bit arrival-time offsets and 5-bit Stream IDs, much less than the size of a packet with header and payload. Details of stream-specific scheduling with a mix of EDF, Fair-share and Priority-class scheduling disciplines is provided in [13].

Concurrency is crucial for Queuing, Scheduling and Data Streaming Figure 8 and Figure 9 show the bandwidth allocations and queuing delay of four streams in the ratio of 1:1:2:4. A key design choice is to allow concurrent queuing of frames, scheduling and streaming. This is done by synchronization-free circular queues with separate read and write pointers as shown in Figure 3. This allows frames to be queued while scheduling decisions and transfer to the network are being completed concurrently. For Figure 8, we report the output bandwidth of streams without making any network stack system calls. We transferred 64000 16-bit packet arrival times from each of the four queues. The zig-zag formation in Figure 9 is because of the traffic generator, which introduces a multi-ms inter-burst delay after the first 4000 frames. Note that the reduced delay for Stream 4 is consistent with Figure 8.

Stream aggregation is easy to achieve using processor resources and the max-finding configuration For best-effort streams that require bandwidth allocations without any requirements for per-stream QoS, aggregation can be achieved quite easily in ShareStreams. If per-stream QoS is desired then mapping each stream to a Register Base block is essential. If aggregate QoS over a set of streams meets the requirements, many *streamlets* can be mapped to a Register Base block or *Stream-slot*. The idea is to save FPGA resources for streams not desiring per-stream QoS by using cheaper processor/memory resources. For demonstrating this, we assigned 100 streamlet queues to each stream-slot and measured the bandwidth at the Stream processor as in Figure 8 and graph the results in Figure 10. We simply used a round-robin service policy on the Stream processor between streamlets. This can be done very efficiently on the Stream processor by cycling

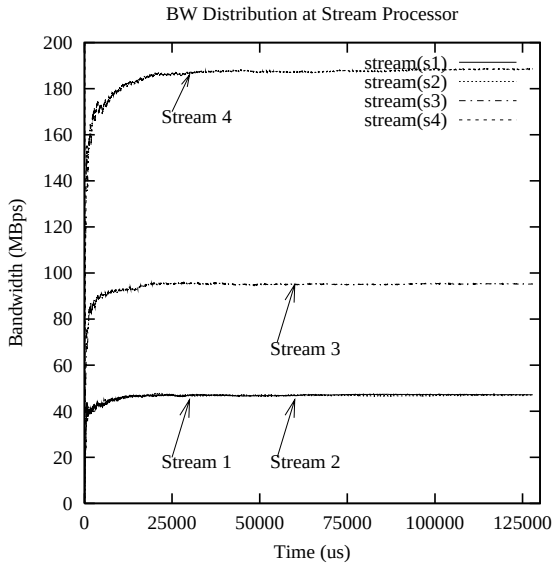

Figure 8: Fair Bandwidth Allocation of Streams (1,2,3,4) with ratios - 1:1:2:4

through active queues. We were even able to support multiple sets of streamlets within a stream-slot. In Figure 10, stream-slots are divided in the ratio 1:1:2:4 ie. 2.0, 2.0, 4.0 and 8.0 MBps with 100 streamlets in each slot with equal bandwidth allocation. We plot the bandwidth of a streamlet in each stream-slot, with streamlets from each of the two sets plotted for Stream-slot 4. Stream-slot 4 has two streamlet sets, set 1 with double bandwidth than set 2. Round-robin service policy can be completed fast and efficiently on the Stream processor, while more complex ordering and decisions are accelerated on the FPGA.

Evaluation Summary and Extension of Results Block Decisions are beneficial for deadline-constrained real-time streams and do not consume excessive FPGA resources. Block decisions can be useful for priority-class scheduling disciplines as the the relative priority between queues remains constant. For service-tag based fair-queuing disciplines, if the computed *finish-time* of a new packet is higher than the those of the elements in the block, the block can be used for transmission in future packet-times, otherwise the queues will need a re-sort again. If the priority assignment engine assigns monotonically increasing priorities across all streams then block decision can be leveraged to increase scheduler throughput. Max-finding configuration of the architecture is critical for bandwidth allocation and shows better area-clock scaling. Scaling the number of streams when per-stream QoS is not essential but acceptable over a fixed collection of streams, can be performed efficiently in ShareStreams. This is by saving FPGA state storage and using cheaper processor resources instead. QoS is provided at a coarser granularity to achieve scale in a cost-effective fashion.

5.2 Performance Comparison

We compare the performance of the ShareStreams architecture with contemporary systems from industry and academic research projects for both the line-card realization and the Endsystem/host-based router realization.

Endsystem/Host-based Router Configuration

A number of research projects have built software-based routers, not limited to [11, 19, 5]. The Click modular router described in [11] can forward packets at the rate of 333,000 64-byte packets/second and

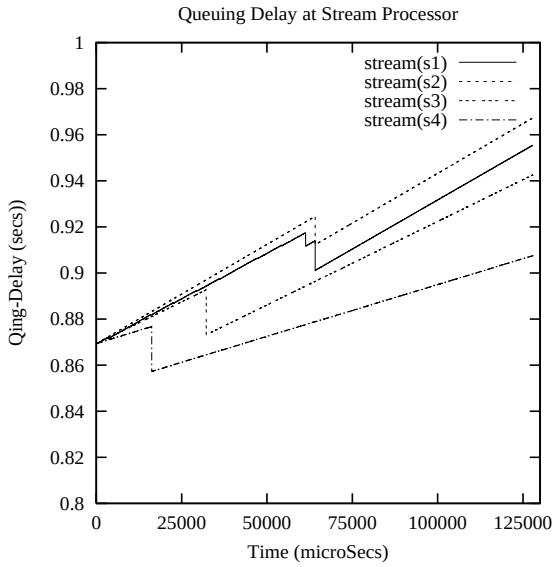

Figure 9: Queuing Delay of Streams 1, 2, 3 and 4

close to 300,000 packets/second with the Stochastic Fairness Queuing(SFQ) module using Linux 2.2 with a 700MHz Pentium III. In ShareStreams, with a Virtex I FPGA, the scheduler throughput with four stream-slots is 7.6 million packets/second in the switch line-card realization shown in Figure 2. This does not include any software overhead of exchanging packet arrival-times and Stream IDs between the FPGA and the Stream processor. This captures the situation where packet arrival-times are supplied in dual-ported memory by action of the switch fabric. When used in the Endsystem/host-based router configuration (Pentium III 550MHz Linux 2.4), the performance of the system is 469,483 packets/second. The measurements are similar to those described with Figure 8. We start the clock after 64000 packets from each stream are queued. We do not include the PCI transfer time of arrival-times to the FPGA and stream IDs from FPGA to Stream processor. Also the playout of frames during delivery does not include any socket system calls. If PCI transfer times are included then the throughput is close to 299,065 packets/second (using PCI PIO transfers rather than DMAs). This is comparable to the performance of the click router without construction of a highly-efficient transfer engine for PCI transfers. Similarly [19] describes transfer rates of close to 300,000 packets/second with Linux and 450 MHz Pentium II with different configuration and QoS policies. [5] describes transfer rates of 28,279 packets/second with a Pentium Pro machine and NetBSD 1.2.1 using a router plug-in architecture. Highly efficient host-based router and endsystem realizations can be constructed since the bandwidth required for exchange of arrival-times and stream IDs is a fraction of the bandwidth required for streaming frames across a system. In our prototype the Celoxica card has a SRAM bank which needs to switch ownership between FPGA and Stream processor each time a transfer is made, which is generally the bottleneck for high-performance PCI transfers and this is reported in [25]. We expect peer-peer PCI transfers as described in Figure 3 to enhance the performance, say between a Network Processor on the PCI bus and the Celoxica FPGA PCI card. Similarly data movement from the Stream processor to the network can also limit performance depending on the network stack implementation.

10Gbps Switch Linecards A number of industry products support 10Gbps line-cards, including line-cards for Cisco's GSR 12000 router [21]. The line-card is capable of wire-speed QoS using deficit round-robin (DRR) and Random Early Detect (RED) policies. The line-card supports 8 queues per port. ShareStreams

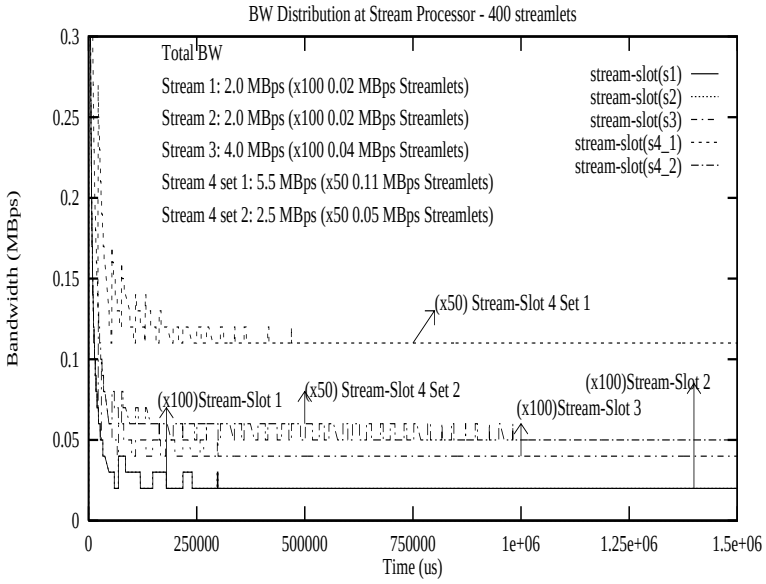

Figure 10: Aggregation of 100 Streamlets into a Stream-slot

can support 32 queues using a single low-end Virtex 1000 chip with more sophisticated DWCS scheduling to meet QoS guarantees required by a mix of real-time streams and best-effort streams. ShareStreams can provide per-flow queuing. The Teracross [20] scheduler chip is built using a Xilinx Virtex II and supports only four service-classes without any per-flow queuing in a 10Gbps line-card configuration.

6 Conclusion and Future Work

This paper presents ShareStreams as a unified canonical architecture for priority-class, fair-queuing and window-constrained scheduling disciplines, along with design-choices and tradeoffs. Using the sorted *block* list of streams is useful for real-time streams with deadlines. This result can be extended to other scheduling disciplines under certain conditions stated in Section 5. The impact on area of routing winners *and* losers for achieving this is found to be minimal as demonstrated in Section 5. Aggregation is useful when per-stream QoS state storage is not essential. Stream-specific deadlines are not possible with aggregation, although the stream-slot they are bound to will be guaranteed a delay-bound. Here we save more expensive FPGA resource state storage area and use cheaper processor memory resources. This is possible because ShareStreams uses a combination of processor and FPGA resources. The performance of the ShareStreams endsystem realization is demonstrated and found competitive with other proposed solutions.

A number of tasks are being completed and are documented in [13], including microarchitectural extensions like *compute-ahead* Register Base blocks that compute state a cycle ahead by using predication, use of hard multipliers in the Xilinx Virtex II architecture to improve performance, demonstration of streaming a mix of real-time streams and fair-share streams and metrics that allow comparison of explored architectures. We are currently integrating those elements of the architecture that will allow us to construct, demonstrate and run a system with hundreds of streams. We hope to relate the implementation information back to the original ShareStreams framework, and it is hoped that this will allow us to construct more customized scheduling solutions (based on traffic types, different scheduling disciplines, cluster configurations and

producer-consumer pairs) that will run at wire-speeds.

Acknowledgments. This work was supported in part by the Department of Energy under its NGI program and by the National Science Foundation by hardware-software infrastructure support from Xilinx, Celoxica, Intel, Aldec and Synplicity Corporations. We thank Prof. Ken Mackenzie for allowing us the use of his Sun/Xilinx CAD machines for multi-day iterative hardware synthesis runs and Craig Ulmer for support with the TPIL PCI communications library for high PCI burst bandwidth interaction with the FPGA card.

References

[1] 10 Gigabit Ethernet Alliance. http://www.10gea.org.

[2] Infiniband Trade Association. http://www.infinibandta.org.

[3] Ranjita Bhagwan and Bill Lin. Fast and scalable priority queue architecture for high-speed network switches. In *INFOCOM (2)*, pages 538–547, 2000.

[4] Celoxica RC 1000 (Virtex 1000) PCI card. http://www.celoxica.com/products/boards/index.htm.

[5] Dan Decasper, Zubin Dittia, Parulkar, and Plattner. Router plug-ins: A software architecture for next-generation routers. In *Proceeedings of ACM SIGCOMM*. ACM, Sept 1998.

[6] A. Demers, Srinivasav Keshav, and S. Shenker. Analysis and simulation of a fair-queueing algorithm. In *Journal of Internetworking Research and Experience*, pages 3–26, Oct 1990.

[7] Differentiated Services Working Group. http://www.ietf.org/ietf.charters/diffserv.charter.

[8] M. Hamdaoui and P. Ramanathan. A dynamic priority assignment technique for streams with (m,k)-firm deadlines. In *IEEE Transactions on Computers, April 1995.*, 1995.

[9] Xilinx Virtex I and Virtex II FPGA Platform Solutions. http://www.xilinx.com.

[10] Aggelos Ioannou and Manolis Katevenis. Pipelined heap (priority queue) management for advanced scheduling in high-speed networks. In *Proceedings of IEEE Conference on Communications Helsinki, Finland (ICC 2001)*, pages 2043-2047, June 2001

[11] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, 2000.

[12] Raj Krishnamurthy, Karsten Schwan, Richard West, and Marcel-Catalin Rosu. A network co-processor-based approach to scalable media streaming in servers. In *International Conference on Parallel Processing*, pages 125–134, 2000.

[13] Raj Krishnamurthy, Yalamanchili, Schwan, and West. Sharestreams: Scalable real-time architecture and hardware support for high-speed qos packet schedulers. In *FY2002 Active System Area Networks Technical Report*, July 2002 url: http://www.cc.gatech.edu/ ˜ rk/asan/tr02.pdf.

[14] Raj Krishnamurthy, Sudhakar Yalamanchili, Karsten Schwan, and David Schimmel. The georgia tech asan project. In *(Work-in-Progress Session) CD-ROM Proceedings of the IEEE Conference on High Performance Computer Architecture(HPCA-7), Monterrey, Mexico, Jan 2001*, Jan 2001.

[15] Raj Krishnamurthy, Sudhakar Yalamanchili, Karsten Schwan, and Richard West. Architecture and hardware for scheduling gigabit packet streams. In *in Proceedings of the 10th IEEE Conference on High-Performance Interconnects (Hoti-02), Stanford University, California*, Aug 21-23, 2002 URL: http://www.cc.gatech.edu/ ˜ rk/hoti02.pdf.

[16] Raj Krishnamurthy, Sudhakar Yalamanchili, Karsten Schwan, and Richard West. Architecture and hardware for scheduling gigabit packet streams. In *(Work-in-Progress Session) CD-ROM Proceedings of the IEEE Conference on High Performance Computer Architecture(HPCA-7), Monterrey, Mexico, Jan 2001*, Jan 2001.

[17] Gordon Mair. Telepresence - the technology and its economic and social implications. In *Proceedings of the IEEE International Symposium on Technology and Society*. IEEE, 1997.

[18] Sun-Whan Moon, Jennifer L. Rexford, and Kang Shin. Scalable hardware priority queue architectures for high-speed packet switches. In *Transactions on Computers*, pages Vol. 49, no.11, pp.1215–1227. IEEE, November 2000.

[19] X. Qie, A. Bavier, L. Peterson, and S. Karlin. Scheduling computations on a programmable router. In Proceedings of the ACM SIGMETRICS 2001 Conference, pages 13–24, June 2001.

[20] Teracross Queuing and Scheduling Chips. http://www.teracross.com.

[21] Cisco GSR 12000 router. http://www.cisco.com.

[22] Tammo Spalink, Scott Karlin, Larry L. Peterson, and Yitzchak Gottlieb. Building a robust software-based router using network processors. In *Symposium on Operating Systems Principles*, pages 216–229, 2001.

[23] Ion Stoica, Hui Zhang, and T. S. Eugene Ng. A hierarchical fair service curve algorithm for link-sharing, real-time, and priority services. *IEEE ACM Transactions on Networking*, 8(2):185–199, 2000.

[24] Trivedi, Hall, Kogut, and Roche. Web-based teleautonomy and telepresence. In *SPIE Optical Science and Technology Conference, 2000*. SPIE, 2000.

[25] Ulmer, Wood, and Yalamanchili. Active sans: Integrating computation and communication. In *IEEE International Symposium on High Performance Computer Architecture (HPCA-8). System Area Networks Workshop.*, Feb 2002.

[26] Richard West and Christian Poellabauer. Analysis of a window-constrained scheduler for real-time and best-effort traffic streams. In *Proceedings of the 21st Real-time Systems Symposium. Orlando, Florida*. IEEE, November 2000. Also available at http://www.cs.bu.edu/fac/richwest.

[27] Richard West, Karsten Schwan, and Christian Poellabauer. Scalable scheduling support for loss and delay constrained media streams. In *IEEE Real Time Technology and Applications Symposium*, pages 24–, 1999.

[28] Xilinx. http://www.xilinx.com.

[29] H. Zhang. Service disciplines for guaranteed performance service in packet-switching networks. In *Proc. IEEE, 83(10):1374–96, Oct. 1995.*, 1995.