

# ShareStreams: A Scalable Architecture and Hardware Support for High-Speed QoS Packet Schedulers \*

Raj Krishnamurthy, Sudhakar Yalamanchili, Karsten Schwan

Center for Experimental Research in Computer Systems  
Georgia Institute of Technology

Atlanta, GA 30332

{rk}@cc.gatech.edu

Richard West

Department of Computer Science  
Boston University

Boston, MA

richwest@cs.bu.edu

## Abstract

*This paper presents the ShareStreams (Scalable Hardware Architecture for Stream Schedulers) architecture, hardware and systems software for scheduling gigabit packet streams in cluster server machines and switches. We can provide EDF, Static-priority, Fair-share and DWCS native scheduling support for real-time streams and best-effort streams. 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 new Xilinx Virtex II family easily scales from 4 to 32 streams on a single chip. A running FPGA prototype in a PCI card under systems software control can divide bandwidth of actual data 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 even more interesting 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 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 work-

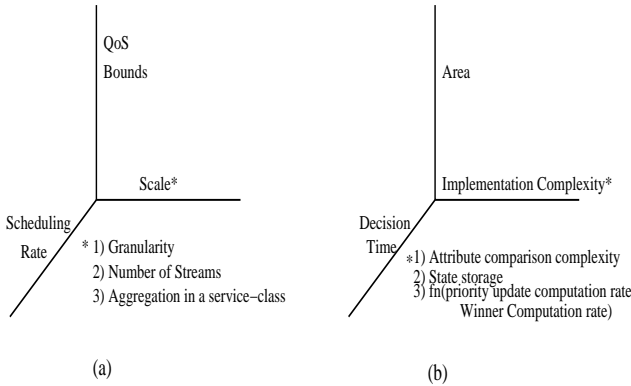
loads. Third, (3) Systems-on-a-chip (SoC) solutions [5] that combine a microprocessor datapath with a reprogrammable logic fabric [5] are now available, as are optimized datapaths for network packet processing and transmission in network processors. Single-chip FPGAs[5] with customized logic capabilities 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 [11], 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* ( $\frac{\text{packet-length(in bits)}}{\text{line-speed(bps)}}$ ), to maintain high link utilization.

These trends call for an architectural framework that allows us to reason about providing packet scheduling solutions, balancing performance & constraints and making tradeoffs required in their physical realization. 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 winner 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 reconfig-

\*This work was supported in part by the Department of Energy under its NGI program, by the National Science Foundation under a grant from Division of Advanced Networking Infrastructure and Research, by hardware/software infrastructure support from Xilinx, Celoxica, Intel, Aldec and Synplicity Corporations.

urable 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 service) scheduling rate can be realized, what will be the degradation in QoS? (and if this is acceptable to applications), if more streams, or smaller packets need to be serviced while the scheduling discipline is in operation. Note that a scheduling discipline may also aggregate entities into a service-class (say multiple virtual circuits into a priority-level) reflecting the degree of aggregation.



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

We propose the SHARStreamS (Scalable Hardware Architecture for Stream Scheduling) shown in Figure 2 and Figure 3 that 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 microsecond (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.

This paper provides performance evaluation of our architecture using the DWCS (Dynamic Window-constrained Scheduling Discipline) [10]. DWCS is a powerful scheduling framework that can be configured to realize most existing schedul-

ing disciplines such as EDF (Earliest-Deadline First), static-priority, WFQ (Weighted Fair Queuing) [4] and also native (deadline and window-constrained dual-attribute) scheduling. We provide a DWCS primer in Section 2 with details in [10]. To demonstrate the versatility of our architecture, we present a FPGA realization of DWCS along with system software support for queue management and transmission that can meet the packet-time requirements of 10Gbps Ethernet links.

**Outline of Paper.** We begin by presenting the control-flow, data-flow and implementation complexity of packet scheduling algorithms in Section 2 and propose a general-purpose ShareStreams architecture for implementing packet schedulers. Section 3 describes the ShareStreams queue management and transmission system software along with hardware architecture for end-systems and line-cards. Xilinx Virtex FPGA architecture and implementation details are provided in Section 4. Performance evaluation of the integrated hardware and software system with synthesized area/delay results and run-time evaluation of actual running hardware is presented in Section 5. We compare other architectures in Section 6 and conclude in Section 7.

## 2 Packet Schedulers: Properties & 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, 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. For comparing multiple service attributes from two streams simultaneously, usually a hierarchy of rules is necessary and an example is listed in Table 1 for the case of DWCS (Dynamic Window-constrained Scheduling). For the purposes of this paper, we call comparing stream attributes from two different streams as a *decision* and a *decision cycle* involves (1) ordering streams based on rules and (2) updating their priorities after *all* the streams are ordered. A decision yields a *winner* and a *loser* stream; after a decision cycle, the stream with the highest priority is said to be a *winner* over other *loser* streams. The scheduling discipline must also ensure that the scheduling decision is completed in a packet-time  $\frac{(\text{packet-length(in bits)})}{\text{wire-speed(bps)}}$  to ensure maximum link utilization. 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. This may be necessary for guaranteeing real-time

bounds (as in an Earliest-Deadline-First EDF scheduler) or allocating bandwidth fairly among best-effort streams. A scheduling discipline must strive to provide performance bounds for real-time and fairness for best-effort streams.

The focus of this paper is to provide an architecture for *dynamic-priority* schedulers that can meet the performance bounds for real-time streams and also provide fairness for best-effort streams. Scheduling disciplines like WFQ[4], SFQ[12], SMART and DWCS[10] are dynamic priority algorithms. In fair-queuing schedulers, [4, 12], a 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[10], deadlines and loss-ratios are used for servicing packets based on rules shown in Table 1. Service attributes of packets are updated every scheduling decision cycle and multiple stream service attributes are used for every *decision*.

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 1. Example Scheduler Decision Rules**

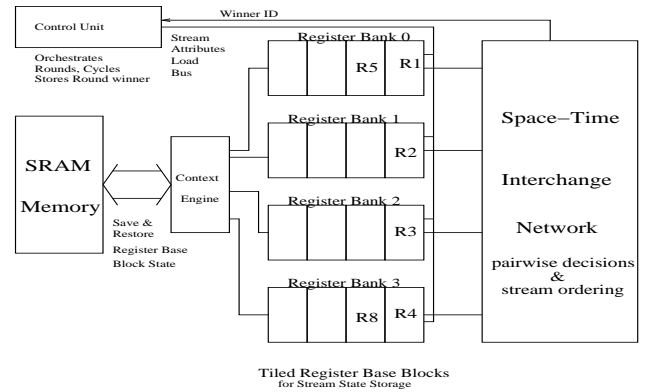
The implementation complexity of a dynamic-priority scheduling discipline is shown in Figure 1 (b) and dependent on the following factors.

*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[4] 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. One way to look at Figure 1 (b) is that the implementation complexity places an upper bound on the scheduling rate. This could be further constrained by area or critical path limitations. A higher scheduling rate might be necessary to maintain QoS bounds or meet scale requirements and it becomes necessary to quantify any possible QoS degradation. Decision time in Figure 1 (b), refers to total time required to pick a winner stream (every

decision cycle).



**Figure 2. ShareStreams Discipline-Independent Hardware Architecture**

**Mapping Packet Scheduling to a Hardware Architecture** Figure 2 presents the ShareStreams architecture for realizing a range of practically interesting dynamic-priority scheduling algorithms. Stream service attributes are stored in each Register base block R1, R2 and also tiled Register Base blocks, like R5 and R8. The space-time interchange network is an appropriate arrangement of Decision blocks (could be simple comparators for single attribute comparison between streams) in a network such as a binary tree, a single-stage recirculating shuffle network, a systolic queue or a shift-register network. *Space-time* interchange network refers to tradeoffs between area (space) and time, allowed by the requirements of a certain architecture. For example, a single-stage recirculating shuffle is a single-level of a binary tree, which can be scheduled in time over  $\log_2(N)$  cycles, and the cycle-by-cycle level traversals form a tree if laid out in space. To save area, one may choose a single-stage recirculating shuffle over a binary tree, and still get full level traversal of a tree over time. The tiled Register blocks store stream state for streams currently being ordered and this allows streams to be scaled beyond a single Register Base block at the base of the network. A winner stream is picked by ordering streams with state in tiled register blocks, and the sequence is termed a *round*. To scale beyond the state in tiled register blocks, state storage from register blocks can be saved in memory and restored when needed during the next decision cycle (after finding a single winner stream across all streams). So assuming there are 32 streams, the winner (or top few, as demanded by the realization) from the first 16 streams will be determined during *round 1*, with state currently available in tiled register blocks. Saved context for the next 16 streams will then be restored into the Register banks from SRAM using the context save/restore engine and the winners computed during *round 2*. Note that the save and restore operations can easily be pipelined as the streams are being ordered. A third round can inter-play the winners from the previous rounds. Note how the architecture captures the dependencies in Figure 1 (a) and Figure 1 (b). To scale the architecture to support more streams, the scheduling rate must be increased to satisfy the QoS bounds or packet-time

requirements of links. We select DWCS (Dynamic Window-Constrained Scheduling)[10] to realize our architecture because it can provide EDF, static-priority, fair-share and native scheduling support for streams using a single realization.

**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 1. The winner stream is then picked for service and its deadlines and loss-tolerances adjusted. We refer the reader to [10] for details but simply state here that the deadline and loss-tolerance adjustments are simple arithmetic operations (increments to deadlines and increments/decrements to loss-numerator and denominator). Similarly, other streams waiting for service have their deadlines and loss-tolerances adjusted in a different manner from the 'winner' stream if they miss their deadlines (in effect to increase their priorities). Streams without any deadline misses are not adjusted. This combination of deadline and loss-tolerance specifications allows DWCS to provide real-time guarantees and fair bandwidth division for streams. In fact, DWCS can be configured to operate as an EDF scheduler (loss-tolerances are  $\frac{0}{0}$ ), static priority scheduler (infinite deadline, static priority is original loss-tolerance of streams) and fair scheduler (WFQ weights can be set using deadlines and loss-tolerances) [10, 11].

The next Section describes the ShareStreams architecture, the software architecture for managing packet queues along with a hardware scheduling discipline realization to schedule streams using the ShareStreams architecture.

### 3 The ShareStreams Hardware & Software Architecture

Certain functions like QoS provisioning are particularly sensitive to temporal bounds and need hardware support and acceleration to satisfy the packet-time requirements of outgoing links. The ShareStreams architecture provides support for data-movement and per-stream queuing on Stream processors and provides hardware support for accelerating packet scheduling in FPGAs.

**The Stream Processor** The ShareStreams architecture maintains per-stream queues usually created on a stream processor (see Figure 3). In Figure 3, per-stream queues are usually created on the host processor (acting as a Stream processor) or Network (Co-)Processor (like the IXP1200) resident on the PCI I/O bus (with specialized hardware units for fast data movement), by a Queue Manager (QM) on stream admission. Our per-stream queues are circular buffers with separate read and

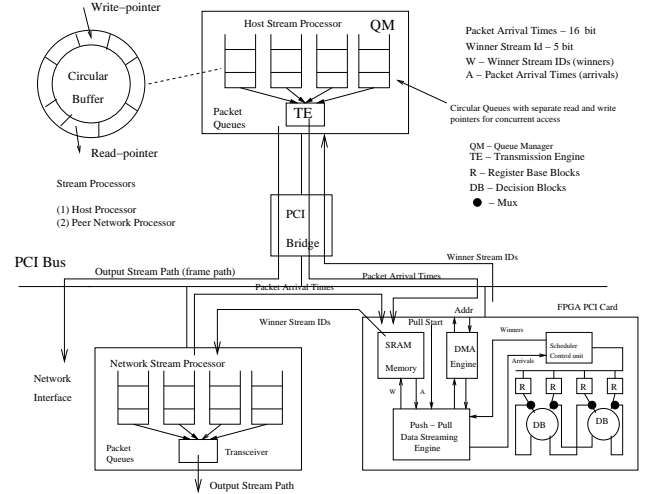


Figure 3. ShareStreams Endsystm Realization

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 stream queues are instantiated, their service constraints are communicated over the PCI bus to a FPGA PCI card by depositing in a SRAM partition (with a special address range). As packets in each stream arrive, their arrival times are communicated to the FPGA PCI card over the PCI bus by direct transfer to Dual-ported SRAM partitions. Most FPGA PCI cards are equipped with banked SRAM that allows multiple stream queues to be accessed concurrently. Transfer of packet arrival times (16 bits, only the offsets are communicated), is usually completed in batch fashion to realize the full burst bandwidth of the I/O interconnect. Note that the *packet arrival times* are only communicated, not the packets themselves. The Scheduler Control unit, accesses packet arrival times from the SRAM with the help of the Streaming Engine in Figure 3. As winners are determined in each cycle, their stream IDs are deposited into the SRAMs to allow reads by Transmission Engine (TE) threads. The Transmission Engine (TE) threads then select the head-of-line packet from *scheduled queues* and transfer the packet to the network interface (usually a DMA pull from the NI). Only 16-bit arrival times are communicated over the PCI bus from the Stream processor to the FPGA PCI card, as are 5-bit scheduled winner stream IDs from the FPGA PCI card to the Stream processor. A line-card realization of the ShareStreams architecture is shown in Figure 4. Dual-ported SRAM allows packets arriving from the crossbar 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 Scheduler control unit.

**ShareStreams Hardware and Streaming Unit** 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 and Decision blocks ar-

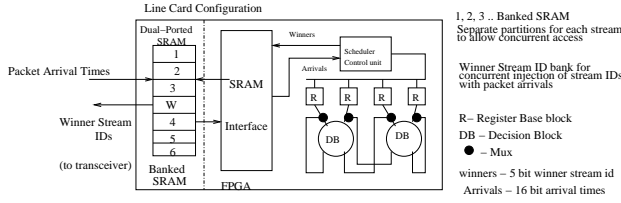


Figure 4. ShareStreams Switch Linecard Realization

ranged in a recirculating shuffle-exchange network (see Figure 5). The Decision block network allows pairwise comparison of streams to determine a winner stream based on current stream service constraints or attributes. Most scheduling algorithms [10, 4], use arrival times for service attribute comparisons and these must be provided to the scheduler hardware during every decision cycle, representing each packet arrival. The recirculating shuffle network generates an ordered list of streams and the winning stream ID is provided to the Control unit which pipelines the IDs to the SRAM interface every decision cycle. The function of the Streaming unit in Figure 3, is to ensure that packet arrivals on the Stream processor are communicated to the Scheduler hardware with low-latency and winner Stream IDs are communicated to the Stream processor with low-latency so that packets in circular stream queues experience minimal delay. The Streaming unit in our design can keep the packet arrival time queue in dual-ported SRAM memory full using a combination of *push* and *pull*. The Stream processor may use PIO writes across the PCI bus (write-combined and bursted on most implementations) to transfer packet arrival times. This is efficient for small transfers, say in the case of an out-of-band time critical stream (see Figure 15) requiring service or a low-bandwidth stream requiring low-delay. For bulk transfers of arrival times (large burst), the Stream processor may request bulk-transfer service using the *pull-start* service line of the Streaming unit (see Figure 3), after setting the registers of the card-resident DMA engine. This allows the Streaming unit to prepare the SRAM partition for bulk-transfers (our hardware implementation uses the Celoxica FPGA card that uses a register-based hardware semaphore to arbitrate SRAM bank access by the host or the scheduler hardware on the FPGA) and pipeline arrival times to the Scheduler Control unit. *Pull-transfers* also allow the Streaming unit to monitor queue levels in SRAM memory and initiate pulls so that the Scheduler control unit always has packet arrival times for winners to be picked and scheduled. *Push-Pull* transfers are also used for transfer of winner Stream IDs from the FPGA processor to the Stream processor Transmission Engine (TE).

**ShareStreams Scheduler Hardware Architecture** The ShareStreams scheduler hardware consists of a push-pull Streaming unit, Scheduler Control unit, Register Base blocks and Decision blocks arranged in a recirculating shuffle-exchange network and is shown in Figure 5. The next Section details the Scheduler hardware architecture and implementation in FPGAs.

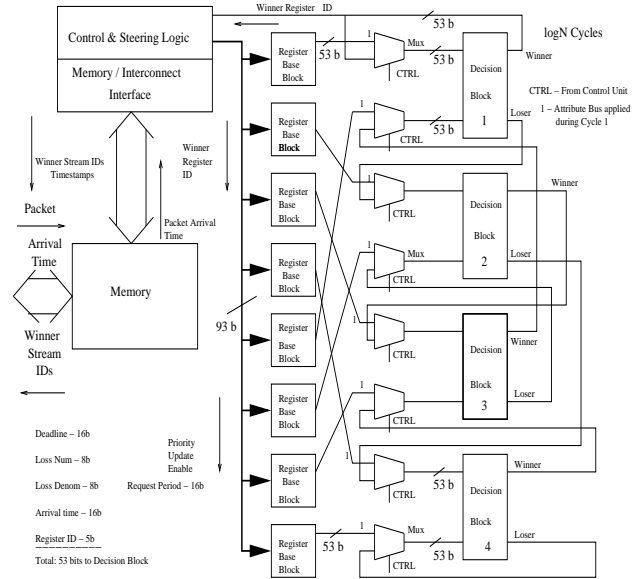


Figure 5. ShareStreams Hardware Architecture: Recirculating Shuffle (All Field Lengths in Bits)

## 4 FPGA Hardware Architecture & Implementation

This Section describes the FPGA hardware implementation of the ShareStreams Scheduler hardware architecture on Xilinx Virtex I & II FPGAs. To demonstrate the versatility of our architecture, we implement a dynamic priority packet scheduling algorithm - DWCS[10], 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.

**Single-Stage Recirculating Shuffle-Exchange Network** As described in Section 3 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. 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.

**Scheduling Timeline** The Scheduling timeline is presented in Figure 6 for an implementation with four streams. During the LOAD cycle, the Register Base blocks are loaded with stream service attributes using the load enable signal driven from the Control & Steering Unit. After the LOAD cycle, the SCHEDULE state and PRIORITY\_UPDATE cycle can begin and will

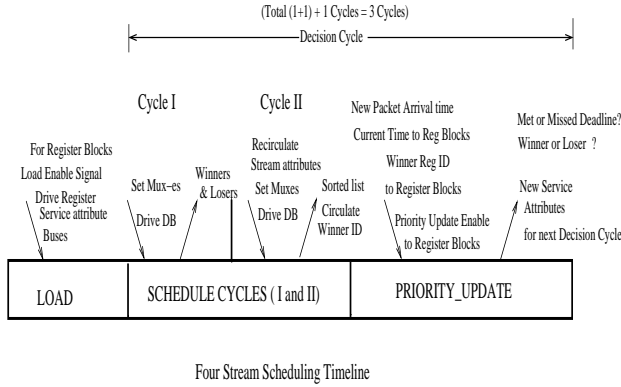


Figure 6. Sharestreams Scheduler Timeline

alternate to generate winner stream IDs. During the I cycle of the SCHEDULE state, the Control unit provides signals for the muxes (see Figure 5 and this allows the stream service attributes to be applied for comparison to the Decision blocks. This will yield winners and losers. The winners and losers from each comparison ie. outputs of the Decision blocks are recirculated during the II cycle of the SCHEDULE state. This allows the winners to be pitched against the winners and the losers to be pitched against the losers, yielding a sorted list of streams after 2 cycles (for a four stream implementation). The winner Stream ID is circulated to the Register Base blocks during the PRIORITY\_UPDATE cycle, along with a PRIORITY\_UPDATE signal and a new packet arrival time for the winning stream and streams that may have dropped a packet. Each Register Base block compares it's ID with with the winner ID, compares the current time (provided by the Control unit) with it's deadline and applies updates (simple arithmetic increments and decrements) based on whether a stream is a winner or a loser and whether it has met or missed a packet deadline.

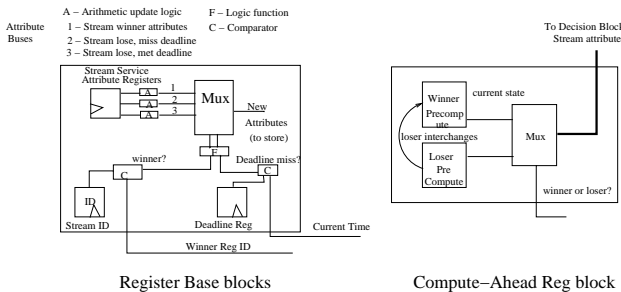


Figure 7. Register Block: Critical Update Logic paths

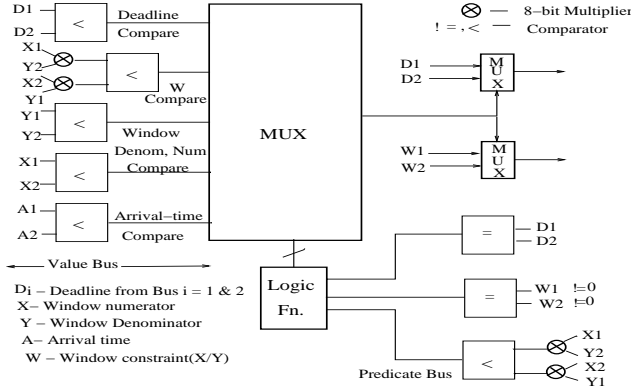
**Control Unit and Register Base Blocks** The Control Unit sequences the recirculating shuffle-exchange network shown in Figure 5 by providing appropriate control signals to sequence the scheduling timeline. The control unit interfaces with the memory controller to provide packet arrival times for Register Base blocks that have scheduled a winner (needs new packet arrival time now) and those streams that have dropped a packet. The Control unit also supplies winner Stream IDs to the mem-

ory interface along with a timestamp. The Control unit maintains a counter and supplies the 16-bit offset timestamp value to the Register Base blocks for comparison with stream deadline values to determine if a packet has missed or met a deadline. The Register Base blocks store Stream service attribute values - individual packet arrival times (16 bit), request periods (16-bit), stream IDs (5-bit), loss-tolerance numerator (8-bit) and loss-tolerance denominator (8-bit), deadlines (16-bit), violation state registers (1-bit), counters (16-bit) and drop packet state storage flag (1-bit). A key element of a Register Base block is a priority update operation which in the case of a dynamic priority algorithm occurs every scheduler decision cycle. For the case of DWCS (see [10]), simple increments/decrements to deadlines and loss-tolerances (numerator and denominator) are applied if the stream is a winner (the Register Base block compares it's ID with the winner stream ID circulated by the Control unit) and similarly, increments/decrements to deadlines and loss-tolerances are applied to loser Register blocks that miss a deadline (a block loses a decision cycle, if the circulated winner Stream ID is different from it's Stream ID). Figure 7 shows the logic operations needed during a PRIORITY\_UPDATE.

**Decision Block** A Decision Block is a key element of the ShareStreams hardware architecture. Figure 8 shows the logic architecture of a Decision block implementing the scheduler rules in Table 1. A Decision block is provided two sets of inputs, usually stream service attributes, representing two streams whose service attributes must be ordered to determine the stream with the higher "priority" ("priority-ordering" must be pre-established using a single service attribute or a combination of service attributes). DWCS uses a combination of service attributes for "priority-ordering" to allow flexible scheduling, which allows service of a mix of EDF, fair-share and static-priority streams. A fair-queuing scheduler might simply use stream weights for stream ordering [12].

Table 1 suggests sequential evaluation of rules that might require multiple cycles for completion by pipelining a comparator chain. This is because of the *apparent* dependencies in the sequential arrangement of rules. For concurrent evaluation of these rules, it is necessary to find independent operations that will allow selection of a winner *predicated* on values of conditions. The idea is to evaluate all possible *compare* operations (cheap operations and only a finite number) and select the relevant operations based on concurrent evaluation of the conditions. One important "compare" operation orders two fractions expressed as numerator and denominator ( $\frac{x_1}{y_1}$  and  $\frac{x_2}{y_2}$ ). We use a 8-bit multiplier ( $x_1 * y_2 == x_2 * y_1$ ) as fast-carry chains and RPM macros are available with the Xilinx Virtex I chip and actual silicon block multipliers are generously scattered across the chip in Virtex II architectures (instead of range limited chordic math). The ShareStreams Decision block for DWCS organizes the rules in simple predicate logic form along a *value-bus* and *predicate-bus*. All the values along the value-bus are evaluated concurrently, but only one is selected by values along the predicate bus. This allows all the rules to be evaluated in just a *single-cycle* and helps lower decision time.

Two architectural variants or modifications are described to the original Base architecture (BA) and termed *Winner-*



**Figure 8. Decision Block: Concurrent Single-Cycle Evaluation**

only Routing (WR) and Compute-Ahead (CA) Register blocks. These are described below.

**Reducing wiring for scaling: Winner-only Routing** Figure 5 suggests how the architecture can be scaled (area, decision-time growth) with increase in stream size. The idea is to route only winners from each shuffle cycle, that need to be recirculated to the next shuffle cycle. Each Decision block only provides winner Stream attribute buses and does not retain ports for loser attribute buses. This can reduce the interconnect requirements and we show the effects on clock-rate and area in Section 5. While area reduction might be marginal as only wires are removed, critical path delay improvement can be substantial as fewer components need be traversed. So for an 8 stream implementation in Figure 5, winners after the first shuffle cycle are pitched against each other. So only Decision block 1 and 3 are used in the second cycle and Decision block 1 in the final cycle.

**Reducing Decision-time: Compute-Ahead Register Blocks** Applications where extra area is not of a concern (large part available or spare area available), *Compute-Ahead* Register blocks can be used to reduce the decision time. The idea is to overlap the SCHEDULE state and PRIORITY\_UPDATE cycle shown in Figure 6. In order to achieve this, a Compute-Ahead Register Base block will compute its updates for a stream being *both* a winner and a loser. These computations are concurrently executed along with the SCHEDULE cycles. As soon as the winner is available after the SCHEDULE state, the stream service attributes (already computed in the previous cycle) can be selected. This *eliminates* a whole cycle and allows successive scheduling decisions to be run without an intervening PRIORITY\_UPDATE cycle and is shown in Figure 7. **Scalability and Decision Time** Our architecture uses  $N$  Register Base blocks for state storage and  $(\frac{N}{2})$  Decision blocks for pairwise packet ordering. A winner is determined in  $\log_2(N)$  cycles of the recirculating shuffle-exchange network and available with a sorted list of streams. Our architecture grows linearly in terms of area and logarithmically in terms of decision time. Service attributes of the streams are usually updated after a winner stream ID is determined. *Compute-Ahead* logic blocks allow trading computation logic area for time, and help determine a winner every

$\log_2(N)$  cycles instead of  $(\log_2(N) + \text{number of cycles for priority update})$ . We show in [11] for a software-based implementation of the scheduler, that it is possible to *relax priority updates* every decision cycle and still maintain bounds without severe degradation. Similarly, scheduling packets in *future packet-times* may be possible using the sorted list of streams every decision cycle i.e. for a set of 8 streams, it may be possible to schedule packets for the current packet-time and 7 future packet-times every decision cycle.

The notions of scalability in ShareStreams are two-fold - *Horizontal* and *Vertical*. To *Horizontally* scale our architecture in Figure 5 from 8 to 16, the number of Register Base blocks are increased from 8 to 16, the number of Decision blocks from 4 to 8 with additional wiring and muxes for control. A winner is available every  $\log_2 16$  or four cycles with this scaling. For *Vertical* scaling, as seen in Figure 2, the Register Base blocks are *tilled* and the Decision block network is *fixed*. So for scheduling 16 streams with *tilled* Register Base blocks for Figure 5, the first 8 streams are scheduled for a winner result after  $\log_2 8$  i.e. 3 cycles and the second 8 streams are scheduled for a winner result in another 3 cycles. This can complete by using two approaches - take another three cycles by choosing the top four streams from each round or complete in just one more cycle by choosing the winner Stream ID from each round. This scaling arrangement is useful for complex Decision blocks requiring a combination of stream service attributes to be compared simultaneously. By fixing the number of Decision blocks and the interconnecting wiring, area savings is achieved by trading time for area (logic implementation and interconnect implementation complexity). This is useful in settings where scheduler cycle times are sub-multiples of packet times i.e. many rounds for winner determination will not exceed packet-times and situations where large granularity packets are being scheduled or as determined by constraints of Figure 1. Note that *aggregation of entities* into a service-class or priority-level is another indirect way of achieving scaling. So in a 32 stream scheduler, we can actually schedule 32 service-classes or priority-levels with tens of virtual-circuits (for example) in each class.

**ShareStreams Scheduler Hardware Prototype** The Control & Steering unit, SRAM memory interface are written in behavioral VHDL (control FSMs) to allow adaptation to new environments. The Decision blocks, Register base blocks and Shuffle-exchange are implemented by wiring components using structural VHDL. These time-critical components use hand-crafted logic by wiring Xilinx corelib components and custom-defined logic components to reduce critical path delay and increase throughput. An evaluation of the architecture with area/delay results is presented in Figure 5. We use the Synplify Pro 7.1 tool for logic synthesis, the Xilinx backend tools (map, place, route) version 4.1 for physical synthesis and bitstream generation. We run the hardware 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 [3].

## 5 Performance Evaluation

This Section evaluates the FPGA hardware implementation and Stream processor software implementation of the ShareStreams architecture. We first present scaling results of the FPGA hardware implementation on Xilinx Virtex I and Virtex II architectures, with architectural variants to improve scalability and delay over the base architecture. The second part of this Section shows the ability of the hardware and software components of the ShareStreams architecture working in tandem to allow stream-specific scheduling ie. EDF, Static-priority (SP) and fair-share (FS) for streams and also for a mix of traffic classes (EDF and FS for example). We demonstrate this for the switch line-card and end-system/host-router target architectures.

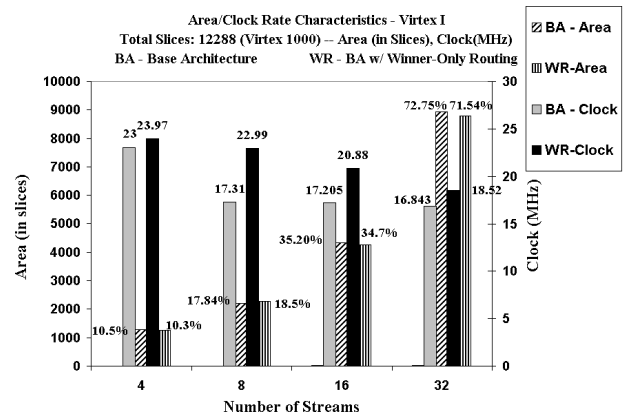
### 5.1 Area-Clock Rate Characteristics

This Section evaluates the ShareStreams base architecture along with two architectural variants discussed in Section 4 - the “Winner-only Routing” (WR) modification and the *compute-ahead* (CA) Register base block. Our design has been targeted to both Virtex I [5] and Virtex II [5] architectures. The Celoxica PCI card has a Virtex 1000 chip and targeting the design to the Virtex I V1000 chip helps us complete run-time performance evaluation. The Virtex II has on-chip block multipliers (hard multipliers that provide single-cycle multiply outputs) which can provide significant gains for scalability and decision time. This Section uses the Virtex II V2000 chip as a synthesis target that has 56 x 48 CLBs (Xilinx II Configurable Logic Blocks) and distinct 56 18\*18 multiplier blocks. A slice includes LUTs and flip-flops and is the basic logic element. Xilinx literature describes a Virtex II CLB equivalent to 4 Virtex II slices and a Xilinx 2V2000 chip capable of a 2 million system-gate design [5]. Similarly, 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).

This Section determines Virtex chip area and decision time (calculated from clock rate, our base architecture can determine a winner in three cycles and our *compute-ahead* register base block variant can compute a winner in two cycles by overlapping priority-updates and winner-selection). The evaluation in this Section does not include the memory/interconnect interface of the design with SRAM memory or any other shared-switched interconnect. Each data-point evaluates the design with Register Base blocks, Decision blocks, control-steering logic block and the recirculating shuffle network. The Figures in this Section use BA to refer to the Base Architecture, WR is the “winner-only routing” architectural variant and CA is the compute-ahead architectural variant with *compute-ahead* Register base blocks. We measured the area of each component in Virtex I and Virtex II and found the area of the Control -steering logic to be 22 slices each (Virtex I and Virtex II), Decision block was 190 slices in Virtex I and 122 slices with 2 additional multipliers under Virtex II. The Register base block was 150 slices in Virtex I and 148 slices in Virtex II. The area of the shuffle-network wires and pass-through CLBs is dependent on the stream size of a given design.

**Area-Decision time Scaling with the Virtex I chip** Figure 9 shows the area-clock rate characteristics of the design from 4 to 32 streams with the Base-architecture and the WR variant. As the number of streams is increased from 4 to 32, the area doubles for the BA architecture. The design stores per-stream state and the state storage doubles at each data-point. The number of Decision blocks needed at each stream size design data-point, also grows linearly, 2, 4, 8 and 16 Decision blocks for stream sizes of 4, 8, 16 and 32 streams. Our architecture grows linearly, in terms of area, and our physically placed & routed designs also exhibit the same rate of growth. For the “winner-only” architectural variant, our Decision blocks only provide outputs of winners, which are recirculated every shuffle cycle. This is expected to ease the routing requirements of the architecture during physical placement and routing. We see some area savings over the base BA architecture.

While area grows linearly, decision time grows logarithmically, exhibiting 2, 3, 4, and 5 cycles for computation of a winner as stream size grows from 4 to 32 streams. To maintain the logarithmic growth, the clock rate must be maintained without any decay and we see from Figure 9 that the clock rate is maintained at nearly the same level between 8, 16 and 32 stream designs for the BA architecture. The change from 4 to 8 streams for the BA architecture shows a 20% drop in clock rate. The WR architecture shows better results over BA in terms of clock rate variation, we see only 5% change in clock rate from 4 to 8 streams and this is nearly maintained over the growth to 32 streams. This can be attributed to easing the physical routing requirements by routing only winners. Reduction in interconnect density can lead to a more compact design with less susceptibility to longer net delays, promoting enhanced clock rate scaling. The “winner-only” routed design variant exhibits better area and clock rate scaling than its base architecture (BA) by reducing physical interconnect requirements.



**Figure 9. Area-Clock Rate Characteristics of Architecture (Virtex I)**

**Area-Decision time Scaling with the Virtex II chip** Figure 10 shows the area/clock-rate characteristics of the Base architecture and the WR architectural variant on the Virtex II chip. The

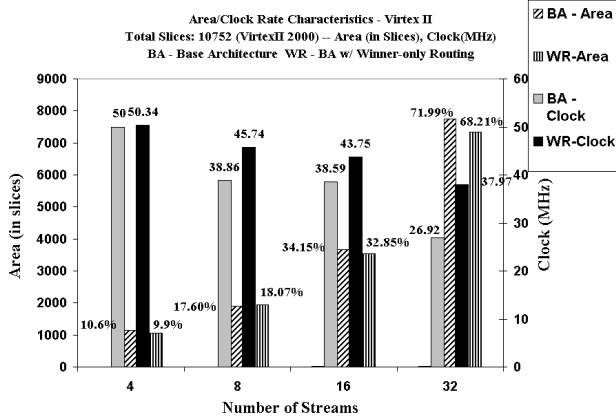


Figure 10. Area-Clock Rate Characteristics of Architecture (Virtex II)

Virtex II has block hardware multipliers (our Virtex I multipliers are relationally-placed macros but need carry-chains which can aggravate net delay) and we expect our design to speedup significantly over our Virtex I implementation. We indeed see this effect, for the BA and WR architectures, the clock rate of the Virtex II designs for all stream sizes are *twice* that of the Virtex I implementation. We see a linear increase in area for the BA and WR architectures, with better area results for WR architecture that only routes winners. The BA architecture clock rate depreciates more than 20% from 4 to 8 streams, and 30% from 16 to 32 streams. The WR architecture, exhibits more stable clock rate variation from 4 to 16 streams with decay of 15% from 16 to 32 streams. *The WR architectural-variant exhibits more stable clock-rate variation than its base architecture and more than double clock rate improvement over the equivalent Virtex I design.*

**Area-Decision time Scaling with Compute-Ahead Register Blocks** *Compute-Ahead* Register blocks are described in Section 4 and shown in Figure 7. *Compute-ahead* Register Base blocks, overlap priority-update with winner selection by pre-computing the new service constraints before the actual winner is available. This allows better decision time as the cycle for priority update is completely removed from the decision time of the packet. This means that a new winner decision cycle can be started after 2, 3, 4, 5 cycles for 4, 8, 16, 32 streams instead of (2+1), (3+1), (4+1), (5+1) cycles. This allows the scheduler to be structured for higher throughput.

The design evaluated in this Section is similar in structure to the BA base architecture, except that the *ComputeAhead* Register Base blocks are larger than the size of a BA Register base block. The *ComputeAhead* Base block calculates and stores state for both the cases of a particular stream winning or losing a decision cycle. When the winner becomes available, the appropriate service constraints are selected and applied for a decision in the next cycle based on whether the Register Base block currently, is a winner or a loser. Figure 11 shows area/clock-rate characteristics for a BA with *ComputeAhead* Register Base

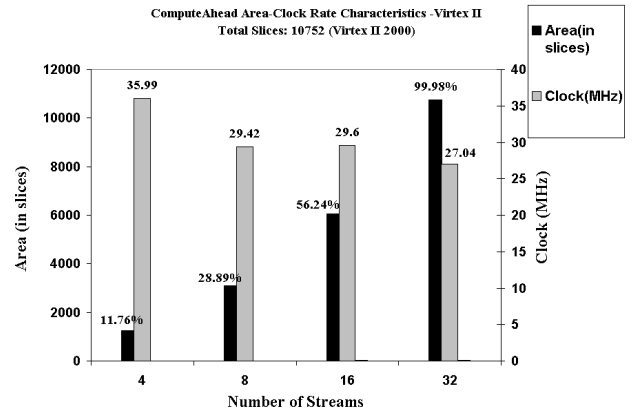


Figure 11. ComputeAhead Register Blocks (BA-based) Results (Virtex II)

blocks. The area grows linearly, from 4 to 32 streams, while clock rate drops 20% from 4 to 8 streams (like the BA architecture in Figure 10) and maintains from 8 to 32 streams.

*Our BA, WR and compute-ahead implementations for both Virtex I and Virtex II 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.* It is also common practice to aggregate many streams into a service-class or priority-class and we can maintain scale by simple aggregation.

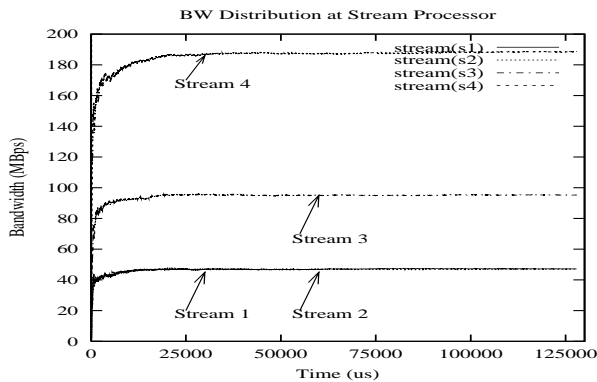
## 5.2 Stream-specific Scheduling

We demonstrate the ability of the ShareStreams architecture to provide stream-specific scheduling for fair-share, EDF and Static-priority streams and also for heterogeneous streams with a mix of traffic classes (FS and SP or SP and EDF). We first demonstrate Stream-specific scheduling for the Stream processor software and FPGA hardware running in tandem for the target architecture shown in Figure 3. We demonstrate customized stream scheduling for EDF, SP and fair-share streams for the target architectures shown in Figure 3 and Figure 4.

**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[3] on a attached PCI bus. The host processor functions as the Stream processor for this target architecture.

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 for each stream are communicated over the PCI bus to the scheduler by writing values into an SRAM partition. Four Streams 1, 2, 3 and 4 are instantiated with window-constraints ( $\frac{7}{8}, \frac{14}{16}, \frac{6}{8}, \frac{4}{8}$ ) i.e. bandwidth-division ratios of 1:1:2:4 (obtained as  $1-\frac{1}{8}$  etc. see [10]). The SRAM interface along with the Control & Steering logic unit shown in Figure 5 provide the required control

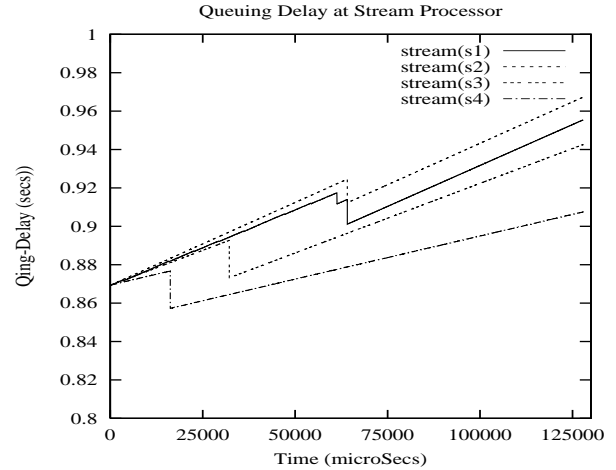
signals to load the service constraint values into the Stream registers. The Transmission Engine (TE) provides arrival times to the ShareStreams hardware scheduler running on the FPGA across the PCI bus and reads winner stream ids and their timestamps from the FPGA PCI card for transmission. Frames are injected by stream producers on the host CPU into per-stream queues resident on the host. Stream producers generate frames between sizes of 64-bytes (null-payload minimum size Ethernet frame) and 1500-bytes (max. transmission unit). This emulates a link-layer Ethernet frame generation system. Each frame producer injects 64000 frames of *varying sizes* with an inter-burst delay interval after the first 4000 frames. The frame producers are allowed to populate the stream queues concurrently and we timestamp each incoming frame. Only 16-bit arrival times are communicated for each packet (in each stream) requiring scheduling to the ShareStreams scheduler running on the FPGA PCI card across the PCI (32bit, 33MHz) bus. Note that the FPGA PCI card also has SRAM banks for buffering incoming packet arrival times and for storing ids of scheduled streams. Extended buffering can also be provided as ends of a data-pipeline from the SRAM banks into the Virtex FPGA on-chip block RAMs. We use the TPIL[7] communications library developed as part of the Active System Area Networks project at Georgia Tech [7]. The FPGA PCI card allows DMAs of over 120 MBytes/second by user-level DMAs from the Celoxica PCI card DMA engines using the TPIL library (peak bandwidth of this bus is 133MBytes/sec). The Transmission engine transferred 64000 arrival-times in each stream to the FPGA SRAM banks (for a total transfer of  $4 \times 64000 \times 16$  bits) and we recorded over 60MBytes/second of transfer bandwidth. After completion of this transfer, the TE pulled winner Streams IDs from the ShareStreams scheduler output SRAM banks for transmission to a remote client. For the purposes of this experiment, we ran the the ShareStreams scheduler at 23MHz on the Celoxica PCI card. We report output bandwidths of each stream by accruing bytes transferred over time at the Transmission Engine (TE) end, without making any network stack system calls.



**Figure 12. Fair Bandwidth Allocation of Streams (1,2,3,4) with ratios - 1:1:2:4**

Figure 12 shows the output bandwidth allocation between Stream 1, Stream 2, Stream 3 and Stream 4. Stream 1 and

Stream 2 settle at an output bandwidth of 47.15 MBytes/second. Stream 3 settles at an output bandwidth of 95.3 MBytes/second, while Stream 4 settles at an output bandwidth of 188.6 MBytes/second. Note that the output bandwidths of Stream 1, Stream 2, Stream 3 and Stream 4 are in the requested ratios of 1:1:2:4.



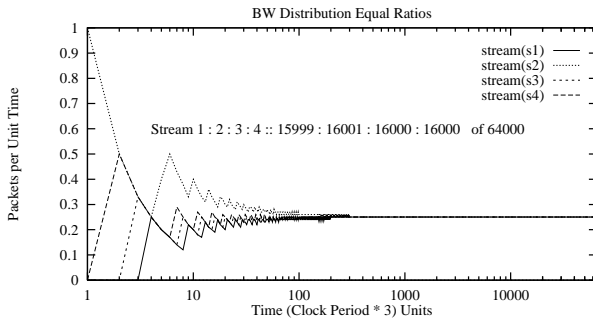
**Figure 13. Queuing Delay of Streams 1, 2, 3 and 4**

Figure 13 shows the queuing delay of the streams at the host processor. Stream producers place frames of varying sizes simultaneously into per-stream queues on the stream processor. Each frame has to wait in queues until the transmission engine begins transmission. This is the total of time taken to generate all 64000 frames per stream queue (there is a 1 ms inter-burst wait period after the first 4000 frames during generation), transfer time to SRAM arrival-time queue buffers on the FPGA PCI card, transfer time of scheduled stream ids from the PCI card to the Stream processor and playout time of all the frames (without any network stack system calls). Note the zig-zag formation in the queuing delay graphs of all streams. The traffic generator introduces a multi-ms inter-burst gap after the first 4000 frames of all streams and all the first 4000 frames in each stream experience increased queuing delay than rest of the frames in the stream. Stream 4 has the highest bandwidth allocation as seen in Figure 12 and incurs the least queuing delay across all frames. Stream 3 has half the bandwidth allocation of Stream 4 and shows slightly more queuing delay across all frames, as Stream 4 frames get serviced more often than Stream 3 frames.

**Stream Scheduling for EDF, SP and Fair-share Traffic Mixes** We demonstrate customized stream scheduling for Earliest-Deadline-First (EDF), Static-priority (SP) mixed with Fair-share traffic for the target architecture shown in Figure 4. For the purposes of these sets of experiments, we deposited service constraints for each stream in the scheduler run-time parameter SRAM partition and backlogged the SRAM partitions for each stream with packet arrivals. We set the clock on the Celoxica FPGA card using our FPGA PCI communications library to 23 MHz, which uses the Virtex 1000 FPGA part. We then started the scheduler and present the outputs of the sched-

uled winners for each traffic mix.

**Fair-share Scheduling for the Line-card Realization** Figure 14 shows the settling of bandwidth of streams with equal division ratios using the line card physical realization described in Section 3. Bandwidth is presented along the y-axis as packets /unit time and time is presented along the y-axis on a log-scale. A bandwidth of 1.0 represents 100% utilization. The time-axis is scaled “real time” - we increment the clock by one tick every time a winner stream is scheduled. A winner is determined every three clock cycles and this makes a time axis tick equal to (clock period x 3). Note that we run the hardware at 23 MHz, which allows us to meet the packet-time requirements of 1 Gbps links. This allows us to verify that winners are available every three clock cycles and enables operation of the scheduler over longer periods of time.



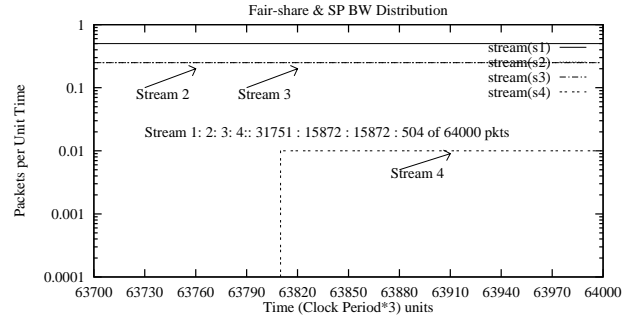
**Figure 14. Fair Bandwidth Allocation of Streams (1,2,3,4) with ratios - 1:1:1:1**

**Deadline-scheduling for EDF Streams** This experiment scheduled four streams - Stream 1, Stream 2, Stream 3 and Stream 4. Each stream had a deadline one-period apart and in strict order 4, 3, 2 and 1. We set the window-constraints/loss-tolerances (numerator and denominator) to be 0 for each stream. Stream ids were equal (15999 each) over a total of 63996 winner stream ids collected.

**Scheduling for a Mix of Fair-Share and Static-Priority Streams**

For this experiment, three fair-share (best-effort) streams are considered along with one static-priority (SP) stream. Stream 1, Stream 2, Stream 3 are set to window-constraints of 2/4, 3/4 and 3/4 i.e. bandwidth division ratios 2:1:1. The static-priority stream is set to a window-constraint of  $\frac{0}{255}$  (no loss of packets allowed over a window of 255 packets) and is set to receive service after 63810 time units. The window-constraint settings of Stream 1, Stream 2, Stream 3 will result in a bandwidth utilization of 100% of the output link together in the specified ratio. After the SP stream is introduced, the window constraint setting of  $\frac{0}{255}$  will not allow other streams to be serviced for a window of 255 packets and only the static priority stream will receive service over that window. Figure 15 plots bandwidth (packets per unit time) on a log-scale against a log-scale time-axis. The time-axis is scaled “real time” - we increment the clock by one tick every time a winner stream is scheduled. Figure 15 plots bandwidth over the final 300 packet window (the scheduler op-

eration terminates after 64000 winners are scheduled). When Stream 1, Stream 2, Stream 3 reach 63700 time units, their output bandwidths have already settled in the ratio of 2:1:1. At 63810 time units, the static-priority stream is introduced and receives service over Stream 1, Stream 2 and Stream 3 as packets over a window of 255 packets are not allowed to be dropped, as specified by the window-constraint. Note that the bandwidth of *all* streams is calculated from time zero (start of playout). The ShareStreams architecture can support out-of-band time-critical traffic in the presence of Fair-share best-effort traffic.



**Figure 15. BW Division of Streams 1, 2, 3 with SP Stream 4**

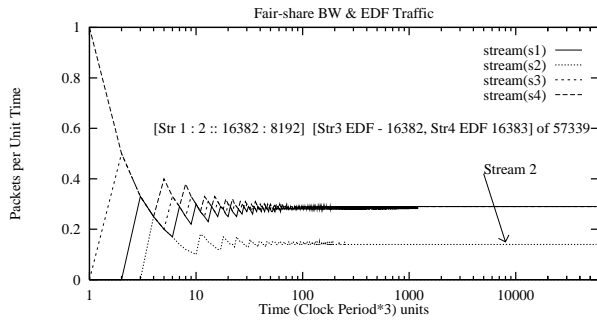
**Scheduling for a Mix of Fair-Share and EDF Streams**

Fair-share best-effort streams can receive service, even in the presence of deadline-oriented EDF streams. Stream 1 and Stream 2 are two fair-share streams with service constraints such that the bandwidth of Stream 1 is twice that of Stream 2. Stream 4 and Stream 3 are two EDF streams that must be serviced one service period apart but, always Stream 4 before Stream 3. Stream 4 and Stream 3 must repeat every four packet-times. Figure 16 shows bandwidth expressed as packets per unit time plotted against time on a log scale. Stream 1 settles receiving twice the bandwidth of Stream 2. The trace was verified to see that Stream 4 was serviced one service-period before Stream 3 always, and repeated every four service periods. The final serviced packet count on Figure 16, shows that Stream 4 and Stream 3 have the same number of serviced packets over the complete set of 57339 packets serviced.

## 6 Related Work

Description and early results from our experience of building a hardware prototype (for a DWCS hardware scheduler) for scheduling upto four streams is described in [8] with Virtex I. While [8] presented early results for bandwidth division, this paper presents a general-purpose architecture for scheduling gigabit packet streams, systems software, scheduling discipline-specific FPGA (Virtex I and Virtex II) implementation with architectural variants, detailed evaluation and scaling results.

A number of hardware structures have been proposed to implement traditional priority queues. [6], [9], [2], all propose interesting priority queuing structures. None of these can be



**Figure 16. Fair-share Streams 1, 2 with EDF Streams 3, 4**

used to target the ShareStreams architecture to realize DWCS packet scheduling efficiently. First, a heap, a systolic queue or a shift-register implementation will require replication of the ShareStreams-DWCS Decision block in every element. The ShareStreams recirculating shuffle conserves area by using only the lowermost-level of a tree. Note that ShareStreams-DWCS 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 re-sorting 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. Successive decision cycles cannot be pipelined through the tree and only one level is operational during a cycle. Instead, a recirculating shuffle conserves area, and *scales* better by using only  $(\frac{N}{2})$  decision blocks in a single-stage recirculating shuffle.

## 7 Conclusion and Future Work

We have described an architectural framework and general-purpose architecture to realize and implement high-speed packet schedulers in multi-gigabit clusters. The tangible outputs of this work include systems software and running FPGA hardware, with design scalability and run-time performance evaluation results in a functional hardware/software system. The ShareStreams-DWCS hardware architecture can easily scale from 4 to 32 streams (or service-classes) on a single Virtex I and Virtex II chip and can meet the packet-time requirements of 10 Gbps links. Tiling register blocks, pipelined context swap/restore engines and bigger single-chip Virtex parts will scale this even better to realize the benefits of per-stream state storage. Our 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. This paper also presents architectural variants that promote scalability (*winner-only routing*) and reducing decision time (*Compute-Ahead*) Register blocks.

Current work is looking at increasing scheduler throughput by

using the sorted list of streams to schedule packets in future packet-times and the impact of this on QoS. The ShareStreams-DWCS FPGA implementation can schedule fair-share, EDF and Static-priority streams. We are currently integrating those elements of the architecture that will allow us to construct, demonstrate and run a system with hundreds of streams. Providing support for other scheduling disciplines is also being considered using the architectural framework and constituent elements. 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.

## References

- [1] 10 Gigabit. Ethernet. Alliance. <http://www.10gea.org>.
- [2] R. Bhagwan and B. Lin. Fast and scalable priority queue architecture for high-speed network switches. In *Proceedings of the INFOCOM Conference 2000 (2)*, pages 538–547, 2000.
- [3] Celoxica RC1000 . Virtex 1000. . PCI. card. <http://www.celoxica.com/products/boards/index.htm>.
- [4] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair-queueing algorithm. In *Journal of Internetworking Research and Experience*, pages 3–26, Oct 1990.
- [5] Xilinx Virtex I and Virtex II FPGA Platform Solutions <http://www.xilinx.com>.
- [6] A. Ioannou and M. 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.
- [7] R. Krishnamurthy, S. Yalamanchili, K. Schwan, and D. 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)*, Jan 2001.
- [8] R. Krishnamurthy, S. Yalamanchili, K. Schwan, and R. West. Architecture and hardware for scheduling gigabit packet streams. In *to appear in the Proceedings of the 10th IEEE Conference on High-Performance Interconnects (Hoti-02)*, Stanford University, California, <http://www.cc.gatech.edu/~rk/hoti02.pdf>.
- [9] S.-W. Moon, J. L. Rexford, and K. 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.
- [10] R. West and C. 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>.
- [11] R. West, K. Schwan, and C. Poellabauer. Scalable scheduling support for loss and delay constrained media streams. In *IEEE Real Time Technology and Applications Symposium*, pages 24–, 1999.
- [12] H. Zhang. Service disciplines for guaranteed performance service in packet-switching networks. In *Proceedings of the IEEE*, 83(10):1374–96, Oct. 1995.