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

Raj Krishnamurthy[†]
IBM Research Zurich Labs
Switzerland
rajk_gt@ieee.org

Sudhakar Yalamanchili, Karsten Schwan, Richard West
Center for Experimental Research in Computer Systems
Georgia Institute of Technology
Atlanta, GA 30332

Abstract

ShareStreams (Scalable Hardware Architectures for Stream Schedulers) is a unified hardware architecture for realizing a range of wire-speed packet scheduling disciplines for output link scheduling. This paper presents opportunities to exploit parallelism, design issues, tradeoffs and evaluation of the FPGA hardware architecture for use in switch network interfaces. The architecture uses processor resources for queueing & data movement and FPGA hardware resources for accelerating decisions and priority updates. The hardware architecture stores state in Register base blocks, stream service attributes are compared using single-cycle decision blocks arranged in a novel single-stage recirculating network. The architecture provides effective mechanisms to trade hardware complexity for lower execution-time in a predictable manner. The hardware realized in a Virtex-I and Virtex-II FPGA can meet the packet-time requirements of 10Gbps links for 256 stream queues with window-constrained scheduling disciplines. The hardware can schedule 1536 stream queues with priority-class/fair-queueing scheduling disciplines using 16 service-classes to meet 10Gbps packet-times.

1 Introduction

Availability of 10GEA (10Gbps Gigabit Ethernet Alliance hardware)[1] and Infiniband-10 Gbps[4] NICs & switches is bringing unprecedented link bandwidth to clusters and long-haul networks. QoS hardware architectures, with packet scheduling disciplines at their core must track the steady growth in wire-speeds to achieve high link utilization and scale. Workloads running on clusters are increasingly a mix of web server traffic, real-time media streams, transaction-processing and scientific workloads. FCFS (First-come-First-Serve) schedulers will easily allow bandwidth-hog streams to flow through and starve other streams, while fair-share schedulers[18] cannot handle real-time media streams with deadlines. On the other hand, sophisticated window-constrained scheduling disciplines running at wire-speeds can meet the diverse QoS requirements of a mix of best-effort *and* real-time streams.

There has been a significant amount of research in the development of packet scheduling disciplines. Results from [17] and [8] show that achievable packet-times with simple (Deficit Round Robin[8]) and sophisticated scheduling disciplines (Dynamic Window-constrained Scheduling (DWCS)[17]) on processors is of the order of 35μs to 50μs. Scheduling disciplines must complete a decision within a packet-time to meet the link requirements ($\frac{packet-length(in\ bits)}{line-speed(bps)}$) of 1 Gbps links (12 μs) and 10Gbps links (1.2 μs) for Ethernet frames. Also forwarding minimum-size packets at line-speeds is a requirement for router line-cards to avoid susceptibility to Denial-of-Service attacks. While previous work has focused on bandwidth, delay and jitter bounds of various algorithms, this paper evaluates opportunities to overlap activities and exploit parallelism in packet scheduling disciplines for wire-speed performance and bound guarantees on FPGA-based architectures. Scheduling disciplines order streams based on stream service attributes and inherently possess a significant amount of bit-level parallelism. A FPGA hardware architecture can exploit this parallelism at the level of a *decision* (pairwise stream ordering) and across all streams for concurrent state maintenance. A tightly-coupled Processor-FPGA systems-on-a-chip (as seen in currently available Triscend A7[5] and Xilinx Virtex II pro [6] families) or standalone FPGA array is a prudent choice to uncover and

[*]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.

[†]This work was completed as a PhD Student at Georgia Tech

exploit this parallelism. A plethora of multimedia applications, systems and protocol software can run on the processor while decisions and stream selection are accelerated by the FPGA array. The processor-FPGA systems-on-a-chip is most suitable for end-systems and low-end host-based routers. On the other hand, a stand-alone FPGA array working in tandem with other chipsets can select streams deposited by a switch fabric in a switch network interface or line-card. FPGA arrays provide a suitable architecture implementation substrate with clock rates up to 400MHz and gate densities up to 10 million gates, supporting low dynamic reconfiguration overheads. The flexibility of software at hardware speeds, available with FPGAs, allows a certain architecture to evolve with the constantly changing landscape of network standards and protocols. A scheduling discipline architecture realized with FPGAs can be customized to evolve with the application, on a dynamic run-time or per-invocation basis. This promotes interoperability and customized operation of scheduling disciplines based on traffic-types, producer-consumer pairs and cluster topologies.

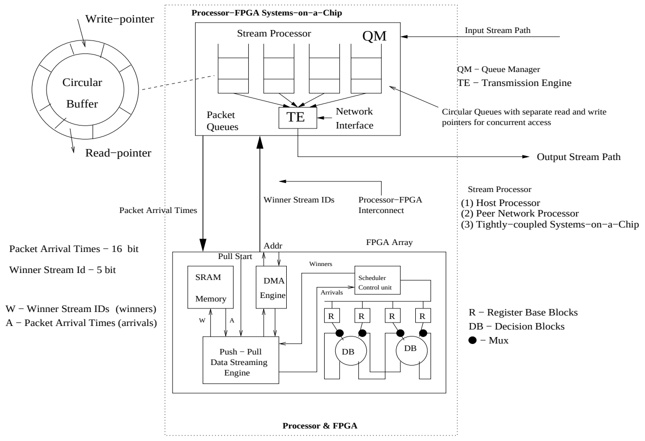


Figure 1. ShareStreams System Architecture: Processor-attached Configuration

This paper describes ShareStreams - a FPGA-based hardware architecture to support a wide range of scheduling disciplines. The ShareStreams architecture allows hardware complexity or area to be traded for lower execution-time in a predictable manner. The focus of this paper is the design issues and tradeoffs in the development of a unified hardware architecture for window-constrained *and* priority-class/fair-queueing (called *service-tag*) scheduling disciplines[10, 16, 9], suitable for use in switch line-cards with output link scheduling. The ShareStreams architecture uncovers and exploits parallelism at the level of a decision and can overlap successive decision-cycles by predication, even for window-constrained scheduling disciplines. The architecture stores state in Register Base blocks, orders streams using single-cycle Decision blocks arranged in a recirculating network to conserve area and improve throughput. Published architectures [15, 7, 11] can support service-tag disciplines, but not window-constrained scheduling discipline operation. We evaluate the tradeoffs required to achieve high scalability and throughput by pipelining the architecture using a Xilinx Virtex II chip [6]. Our architecture can meet the packet-time requirements of 10Gbps links even if scaled to hundreds of stream queues. The design issues and tradeoffs discussed in this paper will apply to a whole range of packet scheduling architectures being developed. This paper describes scaling & pipelining issues in the development of a scalable, high throughput architecture for use in switch network interfaces. Results are provided with Virtex-II and scheduling for a mix of real-time streams & fair-share streams.

2 Packet Schedulers: Complexity & Performance

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, arrival-times) or single-valued (stream weights, start-tags, finish-tags) 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 in [16] for the case of DWCS (Dynamic Window-constrained Scheduling)[16]. For the purposes of this paper, we call comparing stream attributes from two different streams as a *decision* and a *decision cycle* for *winner-selection* involves (1) ordering all streams based on rules and (2) updating their priorities after *all* the streams are ordered. A decision-cycle yields one *winner* and the rest are *loser* streams. The scheduling discipline must also ensure that the scheduling decision is completed in a packet-time $(\frac{packet-length(in\ bits)}{wire-speed(bps)})$ to ensure maximum link utilization. The range of packet scheduling disciplines can be classified as priority-class, fair-share and window-constrained. This paper describes a unified architecture for priority-class, fair-share and window-constrained scheduling disciplines. For priority-class & fair-share scheduling disciplines, the packet priority or service-tag of an inserted packet does not change after being queued, and packets are served in service-tag order. In window-constrained scheduling disciplines, the priorities of enqueued streams is updated every decision cycle for packets in streams that miss deadlines and streams are ordered based on scheduler rules described by us in [16, 13] and shown in Table 1. Priority-class and fair-share scheduling disciplines are collectively called *service-tag* schedulers for the purposes of this paper. Note that *priority assignment* and *winner selection* are the two basic operations common across all scheduling disciplines. Window-constrained scheduling disciplines combine the operation of priority-assignment and priority-update into one cycle, for new packets on arrival (assignment) and packets waiting to be scheduled that miss deadlines (priority update). Window-constrained scheduling disciplines can be regarded as a general case of scheduling discipline operation with multiple degrees of freedom - packet *and* stream-level priority variation. A key insight from this is that priority-class and fair-queueing scheduling disciplines can be realized by bypassing packet-priority updates. This points to the possibility of a unified architecture across a range of packet scheduling disciplines. We show in [12] that window-constrained and service-tag scheduling disciplines exhibit $O(logN)$ complexity when mapped to the ShareStreams architecture described in this paper for N stream queues.

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

Implementation Complexity The implementation complexity of scheduling disciplines is dependent on state storage for service attributes, number of service attributes that are needed for ordering, priority update needs of the scheduling discipline and winner selection rate required to maintain high link utilization.

Concurrency A significant amount of parallelism exists in scheduling discipline operation. Priority assignment and update operations can proceed in parallel across all streams. Pairwise-ordering between any two streams, can proceed in parallel across pairs of streams. Individual conditions for pairwise ordering (a single decision) can be evaluated concurrently. The throughput of a scheduling discipline is limited by the overlap between priority assignment and winner selection. For priority-class and fair-queueing scheduling disciplines, priority assignment and winner selection can proceed completely in parallel. For window-constrained scheduling disciplines, priority assignment/update and winner selection are inherently serialized. We show how this may be overlapped for the case of window-constrained scheduling disciplines in Section 5.

Performance Limits of Processor-resident Schedulers A packet-time of a 64-byte Ethernet frame on a 10Gbps (1Gbps) link is $50ns$ ($512ns$), while that of a 1500-byte frame is 1.2μs (12μs). Detailed performance studies on Sun Ultrasparc 300MHz processors, completed in [17] show that the scheduler latency can be as high as $\approx 50\mu$s for window-constrained scheduling disciplines. Results in [8] 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. A relaxed implementation of DWCS [19] required lazy priority-updates to achieve 1 Gbps throughput with 1K packets using the Intel IXP 1200 network processor in simulation. Each stream required equal share of the bandwidth. Note that lazy priority updates will not help satisfy unequal bandwidth sharing requirements. Software realizations are unable to uncover stream-level parallelism across all streams and bit-oriented concurrency in pair-wise ordering decisions, leading to poor performance. Processor-resident packet scheduling disciplines cannot meet the packet-time requirements of 10Gbps links.

3 Related Work

A number of hardware structures have been proposed to implement traditional priority queues. In traditional priority queueing 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. [11], [15], [7], all propose interesting priority queueing structures and provide ASIC implementations. None of these architectures can be used to provide a unified architecture for priority-class, fair-queueing *and* window-constrained schedulers. First, a heap, a systolic queue or a shift-register chain implementation will require replication of the Decision block in every element, requiring such blocks for every packet. There are N streams and P packets across all streams. The recirculating shuffle in this paper, conserves area by using only the lowermost-level of a tree (requires $\frac{N}{2}$ blocks only). Note that Decision blocks in the window-constrained architecture of this paper 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 unified 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-queueing scheduling disciplines as N packets with service-tags can be ordered in $log_2 N$ cycles, using simple comparators to compare service-tags. An extra priority update cycle is not needed.

4 The ShareStreams System Architecture

The ShareStreams architecture is realized in two separate forms to meet the requirements of two distinct classes of applications. The *processor-attached* configuration for servers & host-based routers and the *switch line-card* configuration for high-end switches and routers.

Processor-attached Configuration The software architecture queue manager (QM) maintains per-stream queues usually created on a stream processor (see Figure 1). The stream processor is a processing element seen in processor-FPGA architectures like Triscend A7[5] and Virtex II-Pro [6]. The FPGA array is a tightly-coupled functional unit or a separate array configured and programmed by the processor across an interconnect. This allows a plethora of existing multimedia applications, protocol stacks and systems software to be run on the processing element. These applications can benefit from hardware acceleration of decisions and stream selection on the attached FPGA array by using systems software running on the processing element. System software issues, detailed interaction with scheduling hardware and related trade-offs is described in [13]. Our 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. The processor interacts with the FPGA array for three main operations - (i) transfer scheduler attributes, (ii) transfer packet arrival-times and (iii) read winner stream IDs. Only 16-bit packet arrival-time offsets or 32-bit service-tags are transferred and also only 5-bit stream IDs are read across the interconnect. A single copy of packets is maintained in packet-store on the Stream processor. Buffering for arrival-times (processor to FPGA) and stream IDs (FPGA to processor) is possible on the processor and SRAM/Block RAM memory on the FPGA array. SRAM/Block RAM memory on the FPGA array buffers packet arrival-times in a per-stream fashion.

Switch Line-card Configuration A line-card realization of the 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. The decision and stream ordering hardware provides scheduled streams IDs to the scheduler control unit.

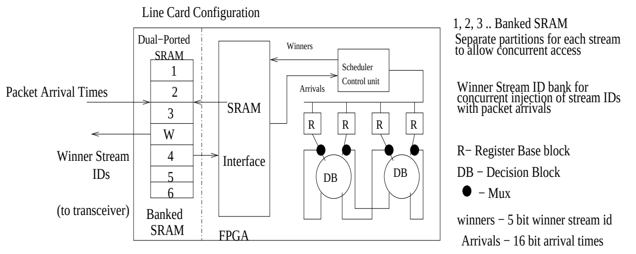


Figure 2. Switch Linecard Configuration

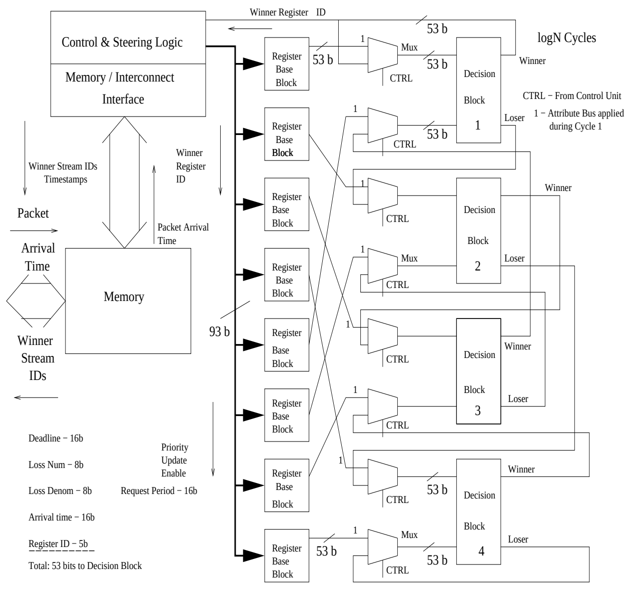


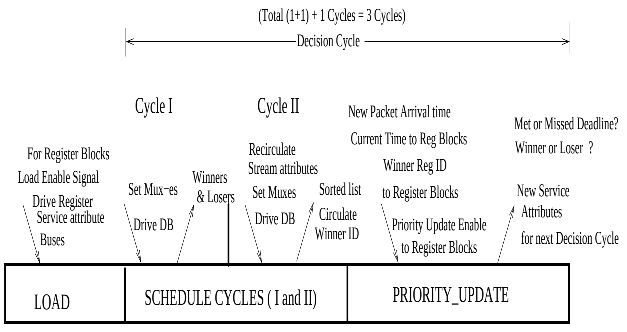
Figure 3. ShareStreams Hardware Architecture: Recirculating Shuffle (All Field Lengths in Bits)

5 ShareStreams Hardware Architecture

This Section describes the FPGA hardware implementation of the scheduler hardware architecture on Xilinx Virtex I & II FPGAs. To demonstrate the versatility of our architecture, we implement *both* a dynamic priority packet scheduling algorithm - DWCS[16] and also a service-tag scheduling discipline with variable number of priority-levels.

Single-Stage Recirculating Shuffle-Exchange Network As described in Section 4 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 $log_2(N) + 1$ cycles for N stream Register Base blocks, using a single-stage recirculating shuffle-exchange network. The winner is obtained after $log_2(N)$ cycles. A sorted list of streams is obtained after $log_2(N) + 1$ 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. 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. The architectural arrangement is shown in Figure 3. When this arrangement is used for service-tag ie. priority-class or fair-queueing scheduling disciplines, an extra PRIORITY_UPDATE cycle is not needed. In this case, two succeeding decision cycles can start without an intervening PRIORITY_UPDATE cycle.

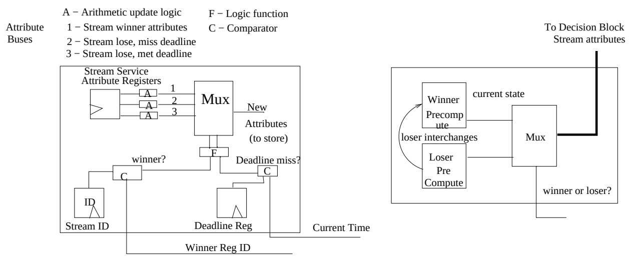
Scheduling Timeline The Scheduling timeline is presented



Four Stream Scheduling Timeline

Figure 4. Sharestreams Scheduler Timeline

in Figure 4 for a window-constrained 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 alternate to generate winner stream IDs. During the I cycle of the SCHEDULE state, the Control unit provides signals for the muxes (see Figure 3) 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 steam 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. For service-tag scheduling disciplines, the PRIORITY_UPDATE cycle is not needed. Loading Register Base blocks with service-tags for successive decision cycles can overlap with winner selection.



Register Base blocks   Compute−Ahead Reg block

Figure 5. Register Block: Critical Update Logic paths

Control Unit and Register Base Blocks The Control Unit sequences the recirculating shuffle-exchange network shown in Figure 3 by providing appropriate control signals (to muxes and other steering components) to sequence the scheduling timeline. The control unit loads scheduler stream service attributes into Register Base blocks during the LOAD cycle, packet arrival-times during each decision cycle and forwards winner stream IDs to memory (with timestamps) for transmission every decision cycle. Another important function of the control unit is to maintain a time base and provide this to each Register Base block, for comparison with stream deadline service attributes. This is to determine if a stream has missed or has met it's deadline during each decision cycle. 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 window-constrained algorithm occurs every scheduler decision cycle. For the case of DWCS (see [16]), 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 5 shows the logic operations needed during a PRIORITY_UPDATE. A Register Base block is called a *stream-slot* if stream-state for a set of streams is time-multiplexed by pipelining. Register Base blocks for service-tag realizations of the hardware architecture in Figure 3, simply store service-tags and Stream IDs in registers.

Decision Block A Decision Block is a key element of the hardware architecture. Figure 6 shows the logic architecture of a Decision block implementing scheduler rules in Table 1 for a window-constrained scheduling discipline. 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). Scheduler rules in Table 1 might suggest sequential evaluation 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. Instead decision blocks in Figure 6, evaluate rules concurrently, by firing independent operations (value-bus) and selecting the appropriate operation based on the satisfying condition (predicate-bus). This is possible because sequential rules are arranged in predicate-logic form. This allows the multiple service attribute compare to be evaluated in just one cycle. One important "compare" operation in the concurrent evaluation of rules, compares two fractions expressed as numerator and denominator ($\frac{x_1}{y_1}$ and $\frac{x_2}{y_2}$). We use a 8-bit multiplier for comparison ($x_1 * y_2 == x_2 * y_1$) 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. This allows fraction compares for all possible values of numerators and denominators instead of limited range provided by chordic math approaches. The 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 outputs along the predicate bus. This allows all the rules to be evaluated in just a *single-cycle*.

A Decision block for a service-tag scheduling discipline will need only a simple comparator to order streams. The complexity of the comparator is a function of the priority-field or service-tag length.

Scalability: Choices and Tradeoffs The notions of scalability in this paper are two-fold - *Horizontal* and *Vertical*. To *Horizontally* scale the architecture in Figure 3 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, the Register Base blocks are *tiled* and the Decision block network is *fixed*. Execution proceeds in rounds of 4 Register Base blocks each with a final 'inter-play' round

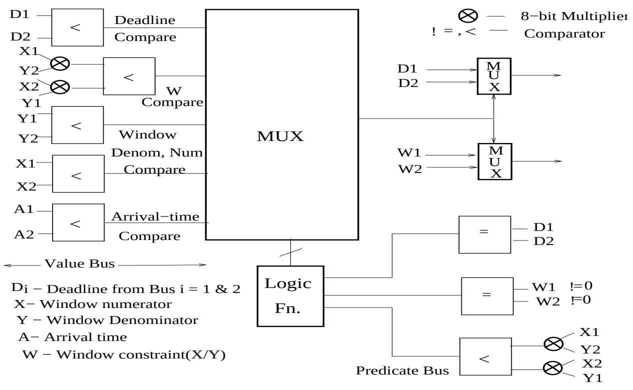


Figure 6. Decision Block: Concurrent Single-Cycle Evaluation with Predicate Logic

to determine the winner across all rounds. Decision-time is logarithmic for each round but an accrued sum over all rounds. This scaling arrangement saves Decision Block network area, by pipelining stream state through a network of reduced size. The tradeoff here is between increased Decision time for reduced Decision Block network area. This might be acceptable for larger granularity packets or slower links, where a small FPGA array is available for decision acceleration. Figure 3 routes both winners and losers to yield a sorted list of streams at the end of $log_2 N$ cycles, termed the *Base Architecture (BA)*. Another approach to improve clock scaling and reduce decision time is to route only winners (*the winner-only (WR) routing architectural variant*). Each Decision block will only output the winner stream, each cycle. So for an 8 stream implementation in Figure 3, all four Decision blocks are used in the first cycle, 1 and 3 in the second cycle and Decision Block 1 in the final cycle. This reduces the physical interconnect requirements, and yields a winner stream which can be recirculated to update priorities. [13] discusses benefits of using the complete sorted list of streams to schedule EDF streams in an efficient *block* manner. We evaluate clock scaling benefits of the WR architecture with Virtex II in Section 6.

Pipelining ShareStreams: Choices and Tradeoffs The hardware architecture shown in Figure 3 for window-constrained scheduling disciplines, orders streams and updates priorities in the succeeding cycle based on whether a stream has won or lost a given decision cycle. By trading increased logic area for reduced decision-time, *Compute-Ahead* Register Base blocks, shown in Figure 5, precompute state for a stream being both a winner or a loser stream. When the winner is selected after a SCHEDULE cycle, appropriate state is selected and used during the next decision cycle without an intervening PRIORITY_UPDATE cycle (shown in Figure 4). Pipelining at the level of a *decision*, allowed by Compute-Ahead Register Base blocks are evaluated in Section 6 (overlaps priority updates and stream ordering by predication). Further pipelining at the level of a given set of streams requiring scheduling support, can be achieved by thinking of Figure 3's Register Base blocks as *Stream-slots*. Such a pipelining scheme can be used for window-constrained as well as service-tag scheduling disciplines. Execution proceeds in *rounds*, with stream state saved and restored from low-latency SRAM memory. After the PRIORITY_UPDATE, an extra cycle is consumed to load Register Base blocks, with stream state restored from memory. This is similar to *Vertical*-scaling without any tiled Register Base blocks. Stream state is time-multiplexed on the hardware by pipelining without using extra Register Base blocks. This can allow many streams (greater than Register Base block count) to be temporally pipelined through a 32-stream slot design, again by trading increased decision-time for reduced state and decision block logic area. Pipelining at the level of a set of streams is possible and is evaluated in Section 6.

Scheduler Hardware Prototype The hardware components were synthesized from VHDL. Time-critical components use hand-crafted logic by wiring Xilinx corelib components and custom-defined logic components to reduce critical path delay. The Synplify Pro 7.1 tool was used for logic synthesis, the Xilinx backend tools 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 up to 100MHz and is equipped with a 32bit/33MHz PCI controller & 8M SRAM accessible from the host and the FPGA[2]. The prototype allows demonstration and verification of scheduling a mix of fair-share, EDF and static-priority streams and is described with experiments in [16, 12].

6 Performance Evaluation

This section evaluates the FPGA hardware implementation of the architecture for window-constrained and service-tag scheduling disciplines. This section presents scaling results, performance comparisons and scheduling issues for a mix of real-time and fair-share streams.

6.1 Window-constrained Architecture

Area-Clock Rate Characteristics This section evaluates the window-constrained base architecture (BA) along with architectural variants discussed in Section 5 - the "Winner-only Routing" (WR) modification, *Compute-Ahead* (CA) Register base blocks and the impact of temporally pipelining stream state. Our design has been targeted to both Virtex I [6] and Virtex II [6] architectures. Decision blocks are efficiently mapped to Virtex II as they use the on-chip silicon block multipliers. Multiplier mappings on Virtex I use logic tree chains and can aggravate net delay.

This section determines Virtex chip area and decision time (calculated from achievable clock rate). 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.

Area-Decision time Scaling with the Virtex I chip

Figure 7 shows area and clock rate achievable with the Base architecture (BA) and the "winner-only" routed (WR) architecture. From Figure 3, the architecture grows linearly in terms of area and logarithmically in terms of decision time. A hardware implementation must attempt to maintain clock rate, with growth from 4 to 32 stream-slots, without any decay, to meet the logarithmic decision time growth. The BA and WR architectures show a linear increase in area from 4 to 32 stream-slots (stream-slots are Register Base blocks of Figure 3). The BA architecture uses slightly more area because of extra wires needed to route loser stream attributes. The WR architecture maintains the clock rate from 4 to 32 stream-slots because of lowered physical interconnect requirements (routes only winners). The BA architecture experiences more logic "spread" because of the need to route winners and losers and shows higher clock rate depreciation than the WR architecture. Our Area/Clock rate results for the BA architecture for Virtex I are from [14].

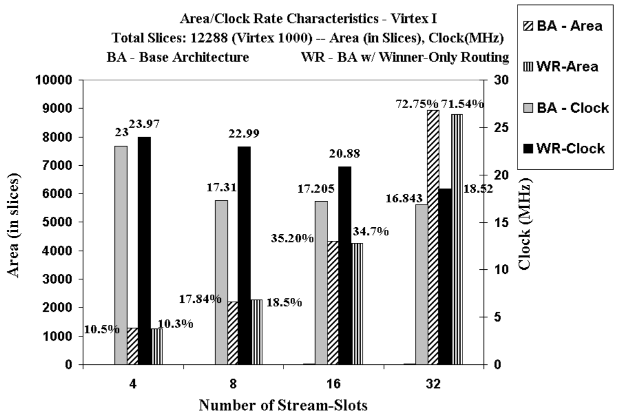


Figure 7. Area-Clock Rate Characteristics (Virtex I)

Area-Decision time Scaling with the Virtex II chip For the results of this paper, we show enhanced performance by mapping to Virtex II. Mapping decision blocks to silicon block multipliers of the Virtex II chip shows a dramatic improvement in clock rate. The hardware architecture on the Virtex II chip uses the block multipliers instead of logic tree chains needed in Virtex I. For both the BA and WR architectures in Figure 8, the clock rate is more than double that of each stream-slot datapoint of Figure 7. Again the WR archi-

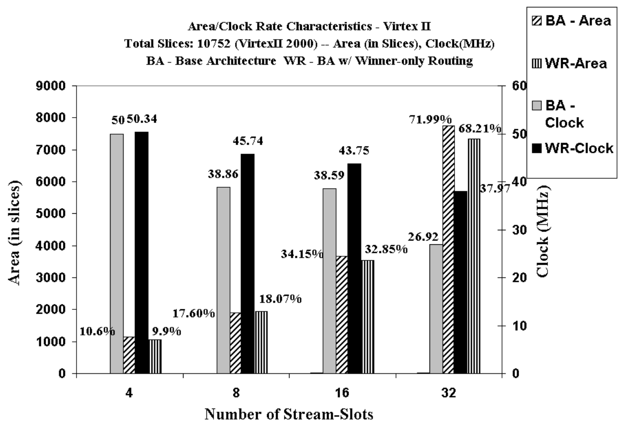


Figure 8. Area-Clock Rate Characteristics (Virtex II)

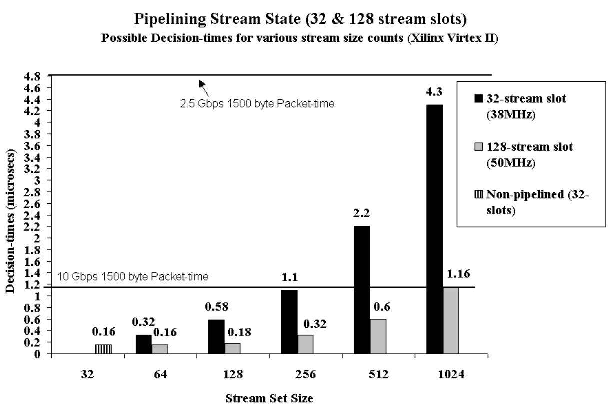


Figure 10. Decision-times with Pipelined Stream state (Virtex II)

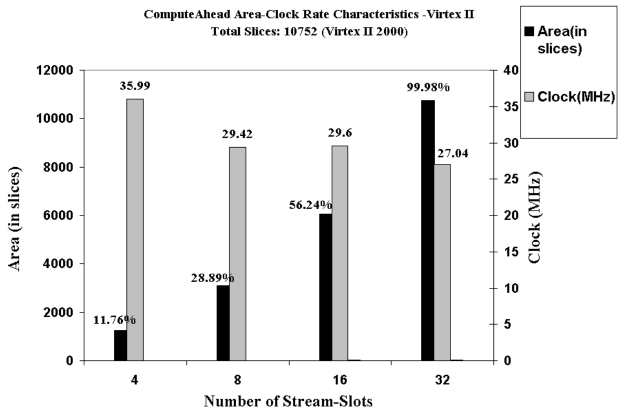


Figure 9. ComputeAhead Register Blocks (BA-based) Results (Virtex II)

tecture preserves the clock rate in a more efficient manner than the BA architecture, because of lowered physical interconnect routing requirements, when growing from 4 to 32 stream-slots. The linear area growth and marginally higher area requirements of the BA architecture (than the WR architecture) of Figure 8 are also consistent with Figure 7. Our Virtex I & II implementations 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. Four stream queues can be serviced at 10Gbps line-rates for 64-byte Ethernet frames with window-constrained scheduling disciplines.

Area-Decision time Scaling with Compute-Ahead Register Blocks *Compute-Ahead* Register blocks are described in Section 5 and evaluated in Figure 9. *Compute-ahead* Register Base blocks, overlap priority-update with winner selection by pre-computing 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. Figure 9 shows the area and clock rate scaling of this architecture. The increased area requirements of this architecture depreciate the clock rate more than the equivalent datapoint in Figure 8 because of logic spread. Each datapoint corresponding to a stream count is still greater than the data-point in Figure 7 for Virtex I. This allows the *Compute-Ahead* architecture to meet the packet-time requirements of 10Gbps links because of fewer cycles needed per decision.

Impact of Pipelining Stream-state on Decision-times The choices and tradeoffs in pipelining the architecture of this paper are discussed in Section 5. Figure 10 shows possible decision times when pipelining stream state through the hardware architecture. If only 32 streams require scheduling support, each one can be mapped to one of the 32 stream-slots and no pipelining is needed. This is equivalent to the 32 stream-slot data-point of Figure 7 and Figure 8. For scheduling more than 32 streams on a 32 stream-slot architecture, stream-state can be pipelined through the architecture. An extra cycle is needed at the end of each decision cycle to save stream state in SRAM memory from Register Base blocks and restore stream state for the next set of streams requiring ordering. Up to 256 streams can be temporally pipelined through a 32 stream-slot design (with a one cycle service attribute load), if packet-time requirements for 10Gbps links are to be met. Up to 1024 streams can be pipelined through a 32 stream-slot design, if packet-times for 1 Gbps and 2.5 Gbps links are to be met. If a 128 stream-slot design were synthesized for Virtex-II (using a bigger part than the Virtex II 2000 used in Figure 8) and clocked at 50MHz, 1024 streams could be pipelined through a 128 stream-slot design to meet 10Gbps packet-times.

6.2 Service-Tag Architecture: Fair-queueing & Priority-class

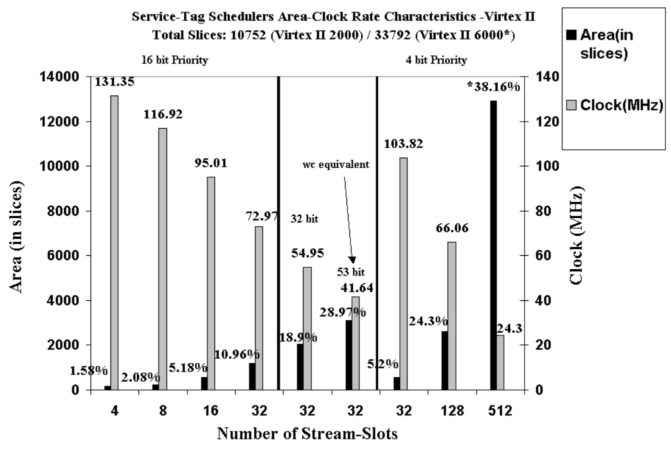


Figure 11. Service-Tag Area-Delay Characteristics (Virtex II)

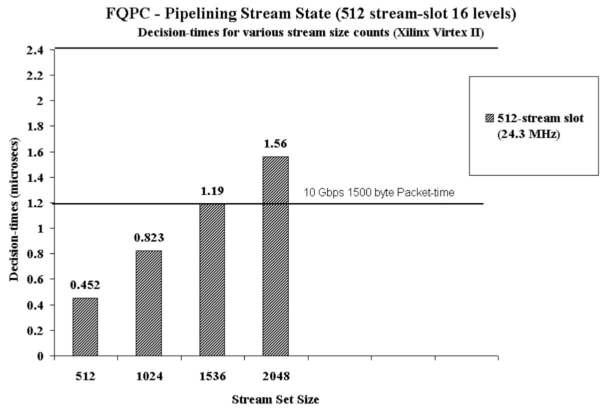


Figure 12. Service-Tag Decision-times with Pipelined Stream State (Virtex II)

Figure 11 reveals the area-delay characteristics of the architecture of Figure 3 for service-tag scheduling disciplines, like priority-class and fair-queueing. Each Register Base block or stream-slot stores service-tags and stream IDs in registers. The Decision blocks use simple comparators arranged in a recirculating shuffle-exchange network. A priority update cycle is not needed. Only winners are routed from the outputs of Decision blocks. Each datapoint evaluates the design with Register Base blocks, Decision blocks, control-steering logic block and the recirculating shuffle network. The service-tag computation engine is not included, as we seek comparisons with the window-constrained architecture. Note that service-tag computation may be completed on the Stream processor in software or directly in hardware, depending on temporal bounds needed by an implementation. Observations regarding linear area growth and logarithmic decision-time growth are similar to the window-constrained architecture of Figure 8. Figure 11 plots area-delay bars for 4 different priority-field lengths. The 32 stream-slot architecture at 4-bit priority-levels routes 9-bit wide wires in the recirculating network. Additional 5-bits are used for addressing 32-stream slots. Similarly, for 16-bit priority-levels, 21 bits are routed and the 32-bit design uses 37 bits for routing and ordering Register Base blocks. The area needs of the 32-stream slot architecture for 16-bit priority-field length are twice that of the 4-bit priority-field length design and approximately half that of the 32-bit design. Given that simple comparators are used for comparison along with registers for storing priority-fields, each additional bit in the priority-field requires an additional wire to be routed. This causes linear growth in area when priority-field lengths are scaled. An important conclusion is that priority-field lengths for service-tag schedulers are a fundamental determinant of hardware complexity in FPGA Virtex II architectures. For 32 stream slots and a 53 bit field length in the service-tag architecture, the achievable clock rate of 41 MHz is close to the achievable clock-rate of the window-constrained architecture of Figure 8. The relevant datapoint in Figure 8 is for 32 stream-slots with winner-routing, that yields a 37.97 MHz clock-rate. This is expected because the window-constrained architecture routes stream service descriptors of 53-bit length. Also, the Decision blocks use silicon multipliers and components that run at rated silicon speed, like comparators in the service-tag decision blocks. This also points to the efficiency of the window-constrained Decision block implementation.

Figure 12 uses a 512 stream-slot design with winner-routing for pipelining stream-state, using a technique similar to the discussion for Figure 10. 1536 stream queues can be supported, while meeting the line-rates of 10Gbps links with 1500-byte Ethernet frames at 16 priority-levels. 32 stream queues can be supported at 16 priority-levels for 64-byte Ethernet frames at 10Gbps meeting 50 ns packet-times. **Evaluation Conclusion** If Decision block components for scheduling discipline realizations use silicon components available on-chip and can be clocked at rated silicon speed, the hardware complexity or area of the architecture implementation in hardware is directly related to the *length of the priority-field or service-classes needed*. 40-byte ACK packets can be supported in a configuration with 16 queues and 16 service-classes for 10Gbps output links. Larger queue sizes can be supported by reducing service classes and aggregating flows.

Performance Comparison

Wire-speed Hardware Architectures [15] gives a comprehensive evaluation of wire-speed hardware architectures for 155 Mbps ATM links for priority-class/fair-share scheduling disciplines synthesized for a 1.2μ CMOS process. The paper provides descriptions of shift-register chains and systolic queues, suitable for mapping service-tag scheduling disciplines, but not window-constrained scheduling disciplines. The binary tree architecture in [15] can be used to map a range of packet scheduling disciplines. For 32 stream queues, the binary tree can complete 13 million dequeue operations per second for 16 priority-levels. For 64 stream queues and 16 priority-levels, around 9 million dequeue operations can be supported. Scaling beyond 64 stream queues is not supported. The hardware architecture described in this paper can complete 21 million dequeue operations per second for 32 stream queues at 16 priority-levels. For 128 stream queues at 16 priority-levels, 9.4 million dequeue operations per second can be supported. The architecture described in this paper scales to schedule 1536 stream queues for 1500-byte packets at 10Gbps.

10Gbps Switch Linecards A number of industry products support 10Gbps line-cards, including line-cards for Cisco's GSR 12000 router [3]. 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. The architecture described in this paper can support 1536 stream queues for 1500-byte packet streams at 10Gbps, and 32 stream queues at 10Gbps for 64-byte packet streams. If more sophisticated window-constrained scheduling is desired for a mix of real-time and fair-share streams, 256 stream queues can be supported at 10Gbps for 1500-byte packets and 4 stream queues at 10Gbps for 64-byte packets. The ShareStreams architecture described in this paper can provide per-flow queueing, with 16 service-classes and 1536 packet queues at 10Gbps for 1500-byte packets. 64-byte packets can be supported at 10Gbps for 32 stream queues with 16 service-classes.

7 Conclusion

FPGA hardware packet schedulers can exploit bit-oriented concurrency in scheduler rules for single-cycle stream pairwise ordering and concurrent state update. This paper develops and evaluates a unified architecture for mapping both window-constrained and service-tag scheduling disciplines on FPGA hardware architectures. The architecture allows area or hardware complexity to be traded for lower execution-time in a predictable manner.

References

[1] 10 gigabit ethernet alliance - http://www.10gea.org.

[2] Celoxica rc 1000 (virtex 1000) pci card, http://www.celoxica.com/products/boards/index.htm.

[3] Cisco gsr 12000 router, http://www.cisco.com.

[4] Infi niband trade association, http://www.infi nibandta.org.

[5] Triscend systems-on-a-chip, "http://www.triscend.com".

[6] Xilinx virtex i and virtex ii fpga platform solutions, http://www.xilinx.com.

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

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

[9] 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.

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

[11] A. Ioannou and M. Katevenis. Pipelined heap (priority queue) management for advanced scheduling in high-speed networks.

[12] R. Krishnamurthy. Scalable real-time architectures and hardware support for high-speed qos packet schedulers. In *Georgia Institute of Technology Doctoral Dissertation*, May 2003.

[13] R. Krishnamurthy, S. Yalamanchili, K. Schwan, and R. West. Leveraging block decisions and aggregation in the share-streams qos architecture. In *International Conference on Parallel and Distributed Processing (IPDPS)*, April 2003.

[14] R. Krishnamurthy, S. Yalamanchili, K. Schwan, and R. West. Architecture and hardware for scheduling gigabit packet streams. In *IEEE Conference on High-Performance (Hot) Interconnects (Hoti-02)*, Aug 2002.

[15] S.-W. Moon, J. L. Rexford, and K. Shin. Scalable hardware priority queue architectures for high-speedpacket switches. In *Transactions on Computers*, pages Vol. 49, no.11, pp.1215–1227. IEEE, November 2000.

[16] R. West and C. Poellabauer. Analysis of a window-constrained scheduler for real-time and best-effort traffi c streams. In *Proceedings of the 21st Real-time Systems Symposium. Orlando,Florida*. IEEE, November 2000.

[17] 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.

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

[19] X. Zhuang, W. Shi, I. Paul, and K. Schwan. Effi cient implementation of a packet scheduling algorithm on high-speed programmable network processors. In *IEEE International Conference on Management of Multimedia Networks and Services MMNS*. IEEE, 2002.