Architecture and Hardware for Scheduling Gigabit Packet Streams*

Raj Krishnamurthy, Sudhakar Yalamanchili, Karsten Schwan and Richard West*

{rk, sudha, schwan}@cc.gatech.edu
Center for Experimental Research in Computer Systems
Georgia Institute of Technology Atlanta, GA 30332-0280

{richwest@cs.bu.edu}
**Department of Computer Science*
Boston University Boston, MA

Abstract

We present an architecture and hardware for scheduling gigabit packet streams in server clusters that combines a Network Processor datapath and an FPGA for use in server NICs and server cluster switches. Our architectural framework can provide EDF, static-priority, fair-share and DWCS native scheduling support for best-effort and real-time streams. This allows – (i) interoperability of scheduling hardware supporting different scheduling disciplines and (ii) helps in providing customized scheduling solutions in server clusters based on traffic type, stream content, stream volume and cluster hardware using a hardware implementation of a scheduler running at wire-speeds. The architecture scales easily from 4 to 32 streams on a single Xilinx Virtex 1000 chip and can support 64-byte - 1500-byte Ethernet frames on a 1 Gbps link and 1500-byte Ethernet frames on a 10 Gbps link. A running hardware prototype of a stream scheduler in a Virtex 1000 PCI card can divide bandwidth based on user specifications and meet the temporal bounds and packet-time requirements of multi-gigabit links.

1.0 Introduction

While core routers carry the Internet's traffic, web and media server clusters at the edges produce and consume heterogeneous traffic streams. These streams require real-time guaranteed services along with best-effort service end-to-end [3, 4, 5]. A stream may originate from a source (possibly a server in a machine cluster) and end at a consumer (another server or display terminal in a machine cluster) located within the same cluster or in another geographically separate location. Also, wire-speeds in server clusters at the edge have been increasing steadily with increasing use of Gigabit Ethernet NIs and switches. Sampling of Infiniband NI chipsets by different vendors brings the promise of Infiniband (2.5Gbps) NI and switch availability to server clusters over the next few months [13, 14]. Flexible scheduling disciplines on the end-system (server machine) and switches are required to meet the real-time and best-effort service requirements of traffic streams delivered by web and media clusters. These scheduling disciplines must operate at wire speeds to match the continual growth of wire speeds in server clusters and maximize link utilization. Use of heterogeneous hardware in the path of a stream with different scheduling discipline capabilities is also possible and must be allowed to promote flexibility, scalability and availability.

Problem Statement

- Scheduling at wire-speeds for optical multi-gigabit links, Infiniband (2.5, 10, 30 Gbps) and the emerging 10GEA (10 Gigabit Ethernet Alliance) [13, 14] standard necessitates scheduling decisions be guaranteed to be completed in a packet time.
- Architecture and implementations are required that can meet cost/performance requirements across a range of environments without ASIC re-engineering overheads.
- It is necessary to trade-off scheduler throughput in packets/sec with quality of service and scheduling granularity, e.g. scheduling at the packet level vs. MPEG frame level.

Solution We address these needs through the use of a powerful scheduling discipline and a flexible target architecture that combines a commercial microprocessor datapath or a Network Processor with a tightly coupled reconfigurable logic component such as a FPGA. Such system-on-a-chip architectures have been announced in the recent past and provide potential hardware solutions for applications that demand both flexibility and the performance that can be achieved via hardware customization [11]. Our approach implements the compute intensive scheduling decision logic within the

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

configurable logic component while control and data movement is handled by the Network Processor. The scheduling discipline for which we propose hardware solutions is Dynamic Window-Constrained Scheduling (DWCS) [2]. DWCS is a powerful scheduling framework that can be configured to implement most existing scheduling disciplines such as EDF (Earliest-Deadline First), static-priority, WFQ (Weighted Fair Queuing) [5] and also native (deadline and window-constrained dual-attribute) scheduling [2]. This could allow heterogeneous schedulers in the path of a stream encompassing end-systems and switches to interoperate and schedule traffic. While DWCS addresses the issue of provisioning QoS, 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. To meet the challenge, we propose a scheduler architecture comprised of a processor datapath coupled with a Field Programmable Gate Array (FPGA). 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.

Precedence among streams for pairwise ordering

- 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. Scheduler Decision Rules

Related Work A number of efforts [15], [10], [8] and [7] have detailed and constructed priority queuing architectures and corresponding hardware in switches and network interfaces. The DWCS scheduling framework allows flexible reconfiguration of the native scheduling discipline into EDF, static priority and fair-share using more complex multiple stream-attribute comparisons. The architectures detailed in [10] are interesting but do not scale well because of the complex nature of our Decision block. The comparator tree requires $\log_2 N$ levels of Decision blocks, while the shift-register and systolic queue architectures require replication of the (comparator) Decision block in each shift-register or systolic block. Our architecture conserves area and scales better by using only log (number of streams) Decision blocks in a recirculating shuffle-exchange network. This is critical in an FPGA architecture where the design must place-and-route and fit on a chip with minimal critical path delay.

2.0 Packet Scheduling : Properties and Implementation 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 descriptors and 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. The scheduling discipline must also ensure that the scheduling decision is completed in a packet-time (packet length in bits / wire-speed of output link) 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 [2, 3, 5].

The DWCS scheduling algorithm is a powerful framework for realizing a range of practically interesting schedulers. Every stream requiring service is assigned two service attributes – a Deadline and a window-constraint or

loss-tolerance (ratio) (W_i) [2]. 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 (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 [2] 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 to increase their priorities). Streams without any deadline misses are not adjusted. These adjustments to deadline and loss-tolerances are simple arithmetic increments and decrements and serve to bias the priorities of streams (increase or decrease their priorities) so that all streams get serviced without starvation during the operation of the scheduling discipline. 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 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) [2, 6]. The reader is referred to [2] where it is also shown that - DWCS can ensure that the delay-bound of any given stream is independent of other streams, delay of service to real-time packet streams is bounded even when the DWCS scheduler is overloaded and also that no more than x packets miss their deadlines every y consecutive packet arrivals, if the minimum aggregate bandwidth requirement of all real-time streams does not exceed the available outgoing link bandwidth. The ability of the DWCS scheduling framework to realize a range of scheduling disciplines for real-time and best effort streams coupled with our experiences with software implementations of the scheduling framework have prompted us to adopt this scheduling framework for architectural exploration and hardware realization[1, 3, 6].

A hardware realization of a dynamic scheduling discipline will need state storage for every stream, pairwise ordering logic for determining the winner stream, priority update logic that updates stream state every scheduling or decision cycle and a control unit to orchestrate progression through the operational states. Our hardware architecture uses Register base blocks (for state storage), Decision blocks (for pairwise stream ordering) and Control & Steering logic with a suitable network as described in Section 3 to realize and allow architectural exploration of a range of scheduling disciplines using a single hardware target architecture.

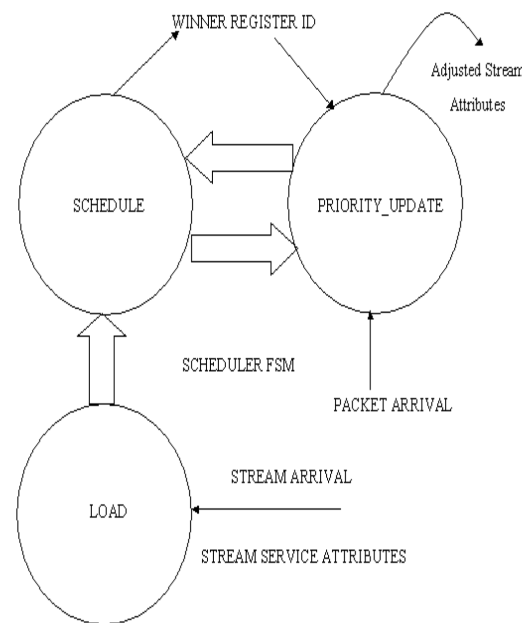

Figure 1. Scheduler Operational States

The principal operations in dynamic packet scheduling will involve LOAD of stream attributes, winner computation by pairwise ordering of stream attributes during a SCHEDULE state and PRIORITY_UPDATE of stream attributes in stream state storage as shown in Figure 1. Once the LOAD operation is complete, the SCHEDULE and PRIORITY_UPDATE operations may alternate. The SCHEDULE and PRIORITY_UPDATE operations are serialized - the winner stream id must be available during PRIORITY_UPDATE for use by each stream to apply the necessary attribute adjustments (the logic in every stream state storage compares its id with the winner stream id). If a Decision block were to order stream attributes pairwise based on the rules in Table 1 then, the Decision block would require deadlines, loss-tolerances (numerator and denominator) and packet arrival times of both streams for multi-attribute

comparison. Table 1 might suggest that each rule needs sequential evaluation and possible multi-cycle Decision block evaluation time. We show in Section 3.2 that sufficient amount of parallelism exists in these rules for concurrent evaluation of each rule resulting in single cycle Decision block evaluation time. A winner must be made available within a packet-time to meet the link rate requirements of the outgoing link. This composition of hardware components (suitably organized to save area and allow scalability) along with minimal operational state transition allows us to provide an architectural solution capable of realizing a range of scheduling disciplines and is detailed in the next Section.

3.0 An Architecture for Realizing Packet Scheduling Algorithms

Network services and their corresponding stacks are frequently realized along the control plane, data plane and management plane dimensions. Control-flow components in control plane (admission control, connection management), data plane and management plane functions are easily mapped to processor instruction sets and datapaths. Dataflow components in data plane functions need acceleration to meet wire-speeds and temporal bounds needed by different layers of the network stack, frequently by exploiting parallelism in the data flow functions. These may be realized as ASICs or preferably in reconfigurable logic or FPGAs to keep pace with the constantly changing landscape of standards and protocols [15, 11].

Our hardware architecture uses a combination of processor datapath (preferably a Network Processor) and FPGA interconnected using a high-speed interconnect. Connection setup, admission control, stream identification and buffering are managed by the processor while packets in streams are scheduled for transmission using the DWCS packet scheduling algorithm realized in the FPGAs. Figure 2 shows packet queues on the stream processor and the hardware scheduler on the FPGA exchanging arrival times (processor to FPGA) and winners (FPGA to processor). The Stream processor may source streams from peer storage entities directly to populate stream queues. The Stream processor software architecture and the FPGA hardware architecture are described in this Section.

3.1 Stream Processor Software Architecture

Stream processor software is usually run on a Network Interface (NI) peer on the same shared, switched or custom interconnect as the FPGA running the scheduler logic.

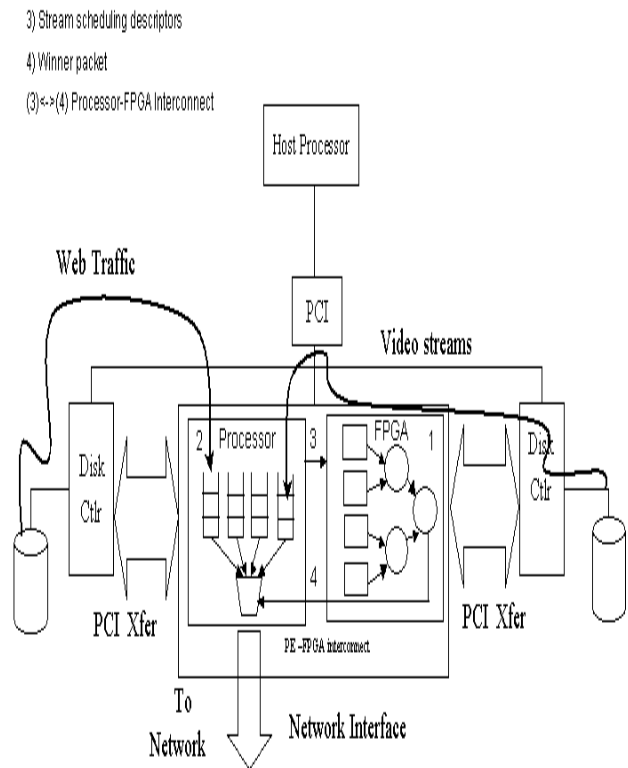

Figure 2. System Hardware and Software Architecture

As streams arrive, input queues are instantiated on the Stream processor. A single copy of packet data is maintained in the Stream Processor and only addresses of frames are placed on input or output queues. Stream Register attributes are loaded into the FPGA stream hardware Registers as streams arrive by communication over the interconnect between the stream processor and state machines running on the FPGA hardware. As packets in each stream arrive, arrival times are communicated for each packet to the FPGA hardware. Packets in each stream are scheduled by the scheduler hardware running in the FPGA and their stream/packet ids are communicated to the stream processor for transmission. Note that an FPGA board and the FPGA chip itself have SRAM banks and on-chip block RAMs to buffer packet arrival times and stream winners (5 bit

stream id) and also that only arrival times (offset or absolute 16-bit values) are actually communicated between the processor and FPGA and winner ids (5-bit) between the FPGA and processor.

3.2 FPGA Hardware Architecture

A hardware realization of QoS packet scheduling will need state storage of stream descriptors and attributes, decision logic for determination of the winner stream every decision cycle and priority update logic for adjusting priorities of streams that got serviced and those waiting for service. The FPGA hardware architecture consists of a Control and Steering logic block, Stream descriptor Register base blocks and Decision blocks. The Control and Steering logic block implements the memory and interconnect interface and provides control and steering signals for the Register base blocks.

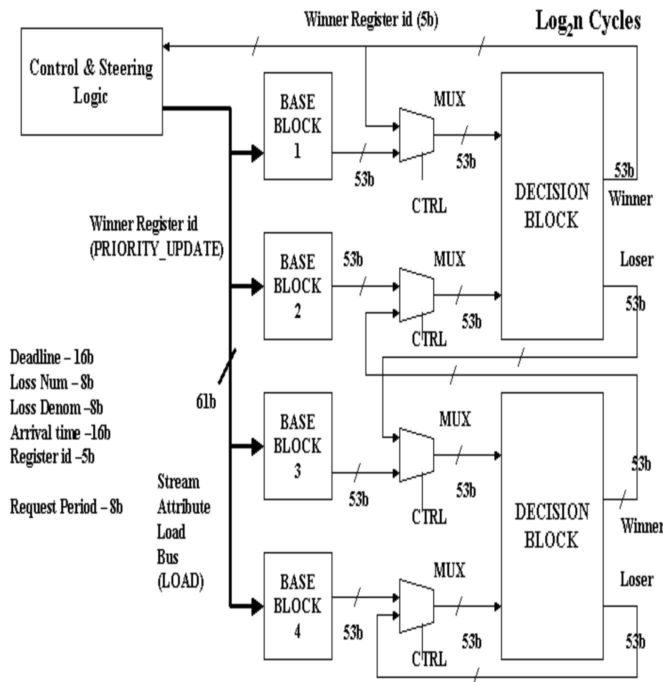

Figure 3. FPGA Hardware Architecture: Recirculating Shuffle-Exchange Network *(Field Length in Bits)*

The Decision blocks are arranged in a single-stage recirculating shuffle exchange network while the Register base blocks form the base of the shuffle-exchange network. The Control & Steering logic blocks, Register base blocks and Decision blocks along with the shuffle network are shown in Figure 3. N streams will require N Register base blocks for state storage, ($\log_2 N$) Decision blocks for winner computation and ($\log_2 N$) cycles of the recirculating shuffle for a sorted list of streams based on stream attributes.

Stream attributes are stored in Register base blocks and stream attributes are updated with logic provided in the Register base blocks. Decision blocks compare stream attribute values between any two streams pairwise and provide winner and loser stream attributes as outputs. A single-stage recirculating shuffle exchange network provides circulation of winners and losers at each stage for a sorted list of streams (based on stream attributes) at the end of $\log_2 N$ cycles. A scheduling timeline usually consists of three states - LOAD, SCHEDULE and PRIORITY_UPDATE after which SCHEDULE and PRIORITY_UPDATE states are repeated as shown in Figure 1.

Stream Register Base Blocks

Register Base blocks store stream attributes – individual packet arrival times (16 bit), request periods (16 bit), stream ids (5 bit), loss-tolerances (numerator (8 bit) and denominator (8 bit)), deadlines (16 bit), violation state registers (1-bit) & counters (16-bit) and also packet-drop flag state storage (1-bit). Stream attribute registers are loaded during a LOAD cycle orchestrated by the Control and Steering logic block in a serial fashion. Arrival times for each packet corresponding to the streams with state in the Stream Register base blocks are updated during the PRIORITY_UPDATE cycle. Stream attribute values are loaded into the registers and the values can be applied to the Decision blocks during the SCHEDULE state. After completion of stream attribute load, the SCHEDULE state can begin and will take $\log_2 N$ cycles for a list of sorted streams along with generation of a winner. At the end of $\log_2 N$ cycles, the PRIORITY_UPDATE cycle can begin and involves only the Register blocks with control signals provided by the Control and Steering logic. At the end of the PRIORITY_UPDATE cycle, a new SCHEDULE cycle can begin with the new stream attribute values available along with the new arrival time of packets corresponding to each Stream Register block. Note that the SCHEDULE cycle and the PRIORITY_UPDATE cycle are serialized. The PRIORITY_UPDATE cycle must complete generating new stream attribute values, before the SCHEDULE cycle can begin.

Decision Blocks

Every scheduler decision cycle, winners are computed by pairwise ordering of streams based on stream attributes. Decision blocks (with degree two) allow

pairwise determination of winner and loser streams which can be recirculated using a shuffle network to obtain a list of streams sorted based on stream attribute values. Stream 'priorities' may be compared based on a single stream attribute (simple comparator) or multiple stream attributes simultaneously. Comparisons are based on 'scheduler rules' (example shown in Table 1) that compare stream attributes and choose the winner stream with the higher 'priority'. An efficient Decision block will compute the results of the 'scheduler rules' and compute a winner stream in a single cycle. The Decision block in our architecture encodes the 'scheduler rules' shown in Table 1 and this may suggest sequential evaluation of each rule, but a careful inspection of Table 1 reveals that each scheduler rule may be evaluated concurrently. We organize the concurrent evaluation of rules as values selected by predicates (just as in a priority encoder) as shown in Figure 4. Logic along all the lines of the value bus are fired concurrently along with all the logic in lines of the predicate bus. Encoding the predicate bus suitably, allows selection of a single bit in the value bus which can be used to select the winner or loser stream. Concurrently allowing 'scheduler rules' to be evaluated based on values selected by their corresponding predicates allows single cycle Decision block evaluation.

Figure 4 shows the value bus consisting of four attribute comparisons (deadline, window constraint, window numerator, window denominator and arrival time). Similarly, the predicate bus has deadline compares and window constraint compares in its datapath. For the purpose of comparing two window constraints, (expressed as a fraction with numerator and denominator) we use an 8-by-8 multiplier to be able to compare over all possible integer values of the numerator and the denominator. For denominators with a power of two, divisions may be implemented as simple shift operations and may also be used along with repeated addition operations for multiplication implementation. We choose to use a more expensive multiplication operator as carry chains are available in the Xilinx Virtex architecture and block hardware multipliers are available in the Virtex II architecture. Additionally, optimal timing for complex operators is available with Xilinx RPM macros and cores.

Recirculating Shuffle Network

Stream Register blocks store stream attribute values while Decision Blocks order streams pair-wise based on stream attribute values. For N streams, the architecture uses $\log_2 N$ Decision Blocks and takes $\log_2 N$ cycles to generate a sorted list of winners based on stream attributes.

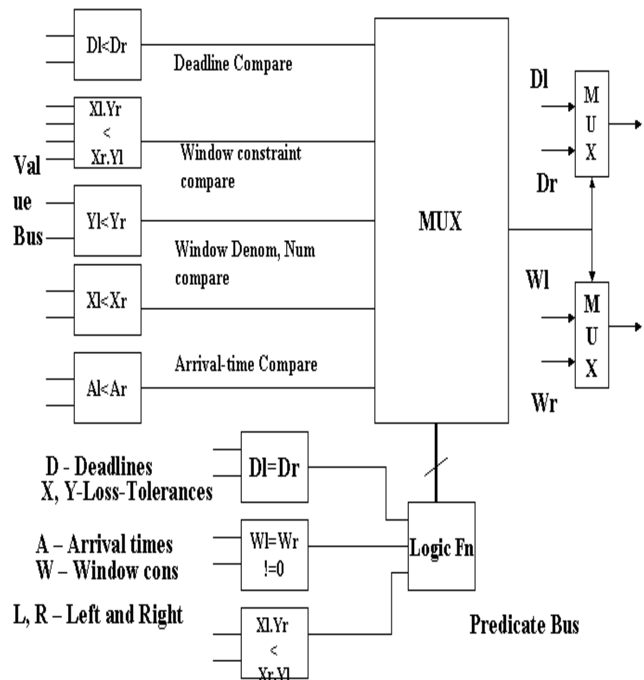

Figure 4. Decision Block Architecture

We utilize a Recirculating shuffle network to accomplish this in $\log_2 N$ cycles for N streams. Consider Figure 3, during the first Decision cycle, stream attributes from stream 1 and stream 2 are applied to Decision Block 1 while stream attributes from stream 3 and stream 4 are applied to Decision block 2. For decision cycle 2, the winners are pitched against the winners and losers are pitched against the losers. Keeping this in mind, the winners from Decision block 1 and Decision block 2 are applied to Decision block 1 and the losers from Decision Cycle 1 are applied to Decision block 2. Muxes at the inputs of Decision block 1 and Decision block 2 apply outputs from the Stream Register blocks or registered outputs of the Decision blocks to each input of the Decision block every cycle. The area in a recirculating shuffle network grows linearly while the scheduling decision time grows logarithmically in the number of streams.

Decision blocks require two attribute buses as inputs (which are 53 bits in width each) and generate winner and loser stream attribute buses as outputs. A recirculating shuffle network temporally schedules the binary Decision block tree (which requires $\log_2 N$ *levels* of Decision blocks) and requires only $\log_2 N$ Decision blocks with a sorted list of winners available every $\log_2 N$ cycles. This saves area, and reduces the number of stream

attribute buses (needed for only one level of the binary comparator tree) which eases placement and routing in an FPGA architecture.

Control and Steering Block

The Control and Steering logic block implements the memory/interconnect interface between the scheduler logic and the Stream processor. As new streams arrive, stream attributes are loaded into Register base blocks by the Control and Steering logic block by assertion of an enable signal provided in each Register base block. As new packets in admitted streams arrive, arrival times in Register base blocks are updated using a different enable signal provided in each Register base block. During a recirculating shuffle, Decision blocks are supplied with stream attribute values from Register base blocks or outputs of the Decision blocks. The Control and Steering logic block provides the necessary controls for the muxes that apply different values for each Decision block during different cycles of the recirculating shuffle. At the end of $\log_2 N$ cycles of the recirculating shuffle, a sorted list of stream attributes is available with the winner stream id provided to the Control and steering logic block for propagation to each Register base block during the PRIORITY_UPDATE cycle. The stream id or register id of each stream is necessary to compute the stream attribute updates based on whether a stream is a winner or loser (has not been selected during the current decision cycle). Similarly, decision of whether a stream has missed or met a deadline is based on comparison of the packet deadline maintained in each Register base block and the current time provided to each Register base block during the PRIORITY_UPDATE cycle.

4.0 Hardware Implementation

A hardware implementation of the above architecture must meet the timing and area needs of the architecture, and we choose an FPGA implementation for flexibility, reconfigurability and enhanced price/performance. The architecture is implemented in a Xilinx Virtex 1000 FPGA using the Synplify Pro 7.0 tool from Synplicity and the ISE 4.1 Xilinx backend tools (map, place, route) from Xilinx. We extensively used the Aldec VHDL simulator for functional and timing simulation of our design [17]. Designs for the Decision blocks, Stream Register base blocks were specified using structural VHDL by wiring up components (from the Xilinx CORE library and our own VHDL component descriptions). Designs for the Control and Steering logic block and memory/interconnect interface were specified using behavioral VHDL and state machines for flexibility and ease of adaptation to new memory/interconnect environments. We synthesized a mapped EDIF netlist using the Synplicity Synplify Pro 7 synthesis tool for different stream capabilities. The EDIF netlist was then placed-and-routed using the Xilinx 4.1i backend tools with various effort levels provided to obtain optimal timing [11, 12]. The netlist was downloaded to a Xilinx Virtex V1000 PCI card for run-time performance evaluation.

For running our hardware design, we used a Virtex 1000 PCI card from Celoxica. The card sports a 32 bit/33MHz PLX controller and 8M SRAM accessible from both a host/PCI peer and the Virtex 1000 FPGA with suitable arbitration (between FPGA and host-PCI peer) provided by the board firmware. Details of the card are provided in [16]. Virtex V1000 bitstreams can be downloaded to the Celoxica card over the PCI bus using a software interface (tool and API) provided by the vendor [16].

5.0 Performance Evaluation

This Section presents the performance evaluation of the architecture and its implementation in Xilinx Virtex V1000 FPGAs [11]. We first present scaling results expressed in terms of FPGA area and maximum clock rate for different stream size capabilities. The running hardware in the Virtex 1000 PCI card is then subjected to stream load for which we report output stream bandwidths for different scheduler attribute specifications.

5.1 Area/Delay Results

For the purpose of this evaluation, we scaled the design to handle a different number of streams from 4 to 32. For each result, we report the FPGA area (in slices) reported by the Xilinx MAP tool and maximum clock rate provided by the Xilinx TRACE tool. Our scalable design specification allows us to scale our designs conveniently by usage of VHDL generics and Figure 5 reports the area/delay results for 4, 8, 16 and 32 streams. We did not include the SRAM memory interface (the control and steering logic was included) for each design datapoint and also did not specify any pin locations so the Xilinx PAR tool could provide the optimal pin assignments to maximize the clock rate. As the number of streams capable of being handled by the design is increased from 4 to 32, the number of slices utilized by the design also grows linearly (doubling at each design data-point). This is expected as a 4 stream design uses four Register base blocks and 2 Decision blocks while the 8 stream design uses 8 Register base blocks and 3 Decision blocks. The

amount of hardware used grows linearly with the number of streams. The maximum clock rate of the 4 stream version is 23 MHz and for rest of the designs the maximum achievable clock rate is around 17 MHz. For slice area utilizations over 10%, the "spread" of logic on

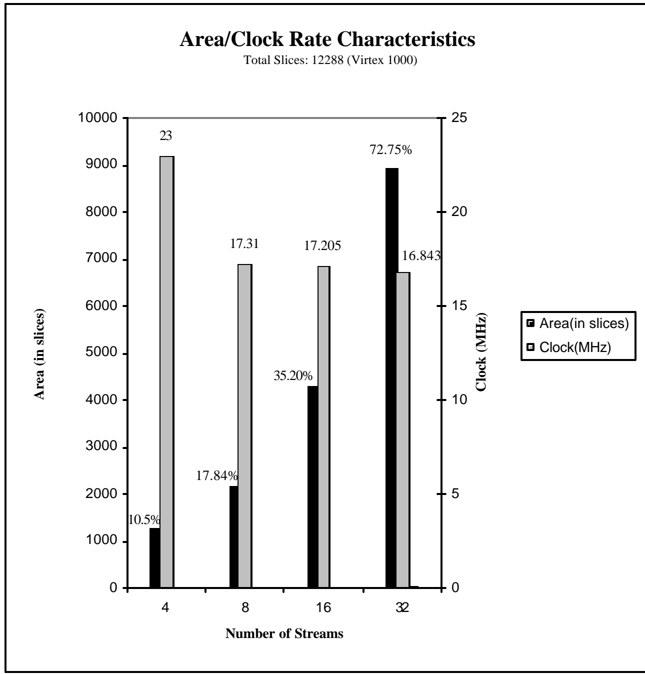

Figure 5. Area/Clock Rate Characteristics

the FPGA increases the critical path delay contributing to the decrease in maximum clock rate. Although 4, 8, 16 and 32 stream designs take $\log_2 N$ cycles ie. 2, 3, 4, and 5 cycles for determination of a winner stream, they all nearly maintain the same critical path delay (observed from the output of the Register blocks to the output of the Decision blocks) to allow nearly the same clock rate. For our architecture, the scheduler area grows linearly but the decision time grows logarithmically. By being able to maintain nearly the same critical path delay, we are able to meet the architecture's theoretical lower bound on decision time growth. Our 4, 8, 16 and 32 stream designs can easily meet the packet-time requirements of 1Gbps (64-byte and 1500-byte frames) and packet-time requirements of 1500 byte frames on 10Gbps links.

5.2 Bandwidth Division

The scheduler operation consists of a LOAD, SCHEDULE and PRIORITY_UPDATE cycle. Once the LOAD cycle is complete, SCHEDULE and PRIORITY_UPDATE cycles are alternated repeatedly. The SCHEDULE cycle yields the winner based on stream attributes and the PRIORITY_UPDATE cycle updates the packet arrival times and stream attributes. The winner stream ids are placed in the SRAM bank during the PRIORITY_UPDATE cycle and packet arrival times are updated from the SRAM during the PRIORITY_UPDATE cycle as well. Appropriate partitioning of the SRAM banks allows concurrent access (17ns access times) for reads (arrival times) and writes (winner ids). Note that stream arrival times and winners are written to and read from the SRAM banks by the Stream processor across the interconnect. For the purposes of this Section, we evaluate our four stream implementation. This Section uses the same design as in Section 5.1 for the four stream version but with an additional SRAM memory interface. Inclusion of the SRAM memory interface does not elongate our clock period and we are able to clock our design at 23MHz as reported in Section 5.1. This allows us to meet the packet time requirements of 1Gbps (64-byte and 1500-byte packets) and 10 Gbps links (1500-byte packets) with ease.

The Celoxica Virtex V1000 board has 4 x 2MB SRAM banks and we partitioned the SRAMs so that winner reporting to SRAMs and packet arrivals could be completed concurrently.

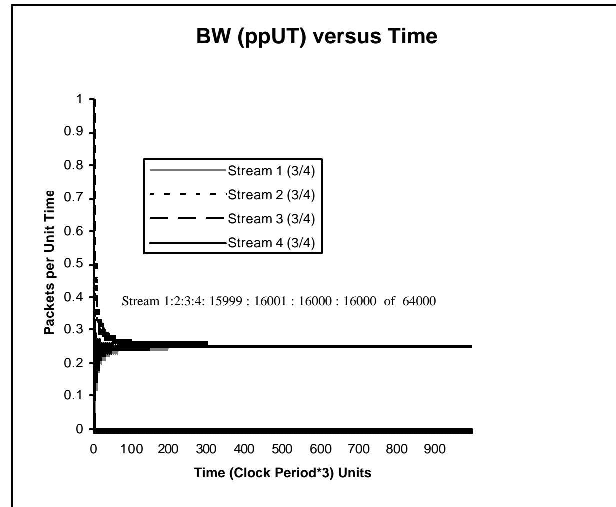

Figure 6. Bandwidth Division results for four streams with equal ratios (1:1:1:1)

We first backlogged the SRAM packet queues with packet arrivals and timestamp values. Scheduler attribute values (Deadlines, loss-tolerances for each stream) were written to an SRAM partition so that Stream Base block registers could be loaded with values when the Control

and steering logic state machine was started. We

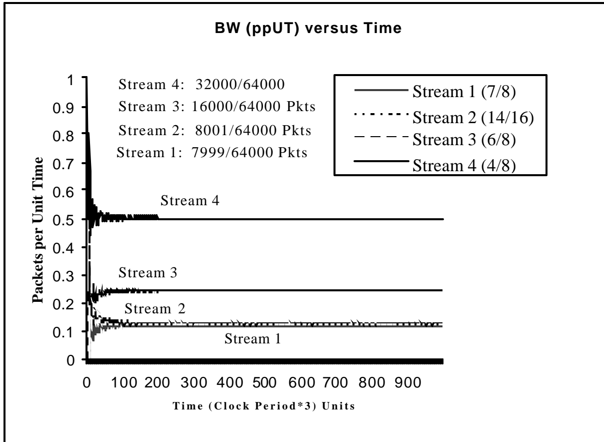

Figure 7. Bandwidth Division results for four streams with ratios (1:1:2:4)

downloaded our design bitstream into the Celoxica Virtex V1000 prototyping card and set the clock to 23 MHz. So during each PRIORITY_UPDATE cycle, winner Register ids are written to the SRAM and packet arrival times loaded for the stream with the recently scheduled packet, along with update of stream attribute values. We performed two bandwidth division experiments. In Experiment I, four streams with loss-tolerances (¾, ¾, ¾, ¾) ie with equal bandwidth division ratios (¼, ¼, ¼, ¼) were scheduled. X-axis reports scheduler output time, we incremented time after each scheduler decision (which deterministically completes *always* in 3 cycles – 2 for the shuffle and 1 cycle for the priority update logic). In other words, this is scaled real-time (clock period x 3 cycles/decision). This allows us to verify that winner outputs are indeed available every three cycles and also allows operation of the scheduler over longer periods of time. We show time from 1 to 1000 time units although the run was terminated after 64000 time units (ie after 64000 winners were scheduled). This allows observation of starting transients and settling time of output bandwidth of each stream clearly. Y-axis reports the bandwidth as number of packets scheduled every unit-time interval (ppUT). The bandwidth of each stream at the output settles after 300 time units and maintains the same value for each stream after the scheduler operation is terminated after 64000 winners are computed. We counted the packets scheduled for each stream and found them to be equal, consistent with the output bandwidth graph. We also verified that winner outputs were available at the end of the PRIORITY_UPDATE cycle, every three system clock cycles, for deterministic temporally predictable scheduling. A second run was also completed and recorded as Experiment II in Figure 7 with loss-tolerances 7/8, 14/16, 6/8, 4/8 ie. bandwidth division ratios (1:1:2:4). The results are shown in Figure 7 and similar arguments as detailed above apply.

6.0 Conclusion

We have proposed and implemented an architecture for exploration of scheduler hardware realizations. The ability of the scheduling framework to support EDF, static-priority and fair-share scheduling along with native (deadline, window-constrained) scheduling allows us to experiment with a range of scheduling disciplines. This allows us to match scheduling disciplines with traffic types, service needs, cluster configurations and different producer/consumer pairs for possible customized scheduling within a cluster. Also, hardware implementing the DWCS scheduling framework can easily interoperate with its predecessor and successor nodes' scheduling disciplines in a stream's path by suitable configuration. The DWCS scheduler stores per-stream state and our architecture grows linearly in the number of streams but logarithmically in terms of decision time. This has been verified by running hardware in an FPGA for 4, 8, 16 and 32 streams. The hardware realization of our architecture for 4, 8, 16 and 32 streams can easily meet the packet times of 1500 byte Ethernet frames on 1Gbps, Infiniband-2.5Gbps and 10Gbps links and all frame sizes on 1 Gbps links. The FPGA hardware implementation for four streams can divide bandwidth among streams in user-specified ratios and also produce winner stream ids. This makes it a viable candidate for end-host scheduling within a cluster and between clusters. Current work is focusing on scalability of this architecture to support a large number of streams in end-systems and cluster switches.

We are looking at architectural and logic changes and tool effort configurations to reduce critical path delay and increase clock rate to meet packet time requirements of all Ethernet frame sizes for 10Gbps links and Infiniband 30Gbps links [13, 14]. We are currently looking at architectural modifications so that the architecture can be completely pipelined for maximum throughput, reduction in interconnect density, reduced area/delay utilization for hundreds of streams and temporal scheduling of the recirculating shuffle network by tiling Register base blocks to support more streams. By retargeting this architecture to the Virtex-II or Virtex2Pro Xilinx FPGA chips, we hope to use the on-chip multipliers and block ram banks to lower critical path delay and increase clock rate to support hundreds of streams.

Acknowledgements We would like to thank the reviewers for their many invaluable suggestions. Our

thanks to Prof. Ken Mackenzie for providing access to his Xilinx CAD machines to allow large scale PAR runs.

References

[1] Raj Krishnamurthy, Sudhakar Yalamanchili, K. Schwan and R. West. Architecture and Hardware Support for Scheduling of Gigabit Packet Streams, Short paper (work-in-progress) in CD-ROM proceedings of the *IEEE Conference on High Performance Computer Architecture(HPCA-7), Monterrey, Mexico, Jan 2001.*

[2] Richard West and C. Poellabauer. Analysis of a Window-Constrained Scheduler for Real-time and Best-Effort Packet Streams. In *Proceedings of the 21 st Real-Time Systems Symposium, Orlando, Florida, November 2000*. Available at http://www.cs.bu.edu/fac/richwest.

[3] Raj Krishnamurthy, K. Schwan, R. West and M. Rosu. A Network CoProcessor-Based Approach to Scalable Media Streaming in Servers, In *Proceedings of the 29 th International Conference on Parallel Processing, Toronto, Canada, July 2000.*

[4] Raj Krishnamurthy, K. Azad, David Schimmel, Ken Mackenzie, K. Schwan and S. Yalamanchili et al, "The Georgia Tech ASAN Project", Short paper(work-in-progress) in CD-ROM proceedings of the *IEEE Conference on High Performance Computer Architecture(HPCA-7), Monterrey, Mexico, Jan 2001.*

[5] A. Demers, S. Keshav and S. Shenker. Analysis and Simulation of a Fair-Queueing Algorithm. *Journal of Internernetworking Research and Experience, pages 3-26, Oct 1990.*

[6] R. West and K. Schwan. Dynamic Window-constrained scheduling for multimedia applications. In *6th International Conference on Multimedia Computing and Systems, ICMCS'99. IEEE, June 1999.*

[7] Srinivasan Keshav. On the efficient implementation of fair queueing. In *Internetworking: Research and Experience Vol.2, 157-173, September 1991.*

[8] J. L. Rexford, A. G. Greenberg, and F. G. Bonomi. Hardware-efficient fair queueing architectures for high-speed networks. *In IEEE INFOCOM'96, San Francisco, March 1996.*

[9] C.L.Wu and T. Feng. The universality of the shuffle-exchange networks, *IEEE Transactions on Computers, vol. C-30, pp 324-332, 1981.*

[10] Sung-Whan Moon, Jennifer Rexford, and Kang Shin. Scalable hardware priority queue architectures for high-speed packet switches. *IEEE Trans. on Computers, vol. 49, no. 11, pp. 1215-1227, November 2000.*

[11] Xilinx Virtex and Virtex-II Chipsets / Xilinx ISE 4.1 Tools. http://www.xilinx.com

[12] Synplicity Synplify Pro 7 Compiler. http://www.synplicity.com

[13] 10GEA Alliance. http://www.10gea.org

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

[15] John W. Lockwood, Jon S. Turner, David E. Taylor, Field Programmable Port Extender (FPX) for Distributed Routing and Queuing, *ACM International Symposium on Field Programmable Gate Arrays (FPGA'2000)*, Monterey, CA, February 2000, pp. 137-144.

[16] Celoxica RC1000 (Virtex V1000) PCI card. http://www.celoxica.com/products/boards/index.htm.

[17] The Aldec VHDL/Verilog/EDIF mixed simulator. http://www.aldec.com