A Router Architecture for Real-Time Point-to-Point Networks

Jennifer Rexford, John Hall, and Kang G. Shin

Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48109-2122
E-mail: {jrexford, hallj, kgshin}@eecs.umich.edu
Web: http://www.eecs.umich.edu/RTCL

Abstract

Parallel machines have the potential to satisfy the large computational demands of emerging real-time applications. These applications require a predictable communication network, where *time-constrained* traffic requires bounds on latency or throughput while good average performance suffices for *best-effort* packets. This paper presents a router architecture that tailors low-level routing, switching, arbitration and flow-control policies to the conflicting demands of each traffic class. The router implements deadline-based scheduling, with packet switching and table-driven multicast routing, to bound end-to-end delay for time-constrained traffic, while allowing best-effort traffic to capitalize on the low-latency routing and switching schemes common in modern parallel machines. To limit the cost of servicing time-constrained traffic, the router shares packet buffers and link-scheduling logic between the multiple output ports. Verilog simulations demonstrate that the design meets the performance goals of both traffic classes in a single-chip solution.

1 Introduction

Emerging real-time applications, such as avionics, industrial process control, and automated manufacturing, impose strict timing requirements on the underlying computing system. As these applications grow in size and complexity, parallel processing plays an important role in satisfying large computational demands. Effective real-time parallel computing hinges on predictable communication between cooperating processing nodes, since lost or late messages can result in human catastrophe or economic harm. However, instead of guaranteeing bounds on *worst-case* communication latency, most existing parallel architectures focus on providing good *average* network throughput and packet delay. Consequently, recent years have seen increasing interest in developing networks that provide performance guarantees for communication in parallel machines [1–6].

Real-time systems employ a variety of network architectures, depending on the application performance requirements. Although prioritized bus and ring networks are commonly used in small-scale systems [7], larger applications can benefit from the higher bandwidth available in multi-hop topologies. In addition, multi-hop networks often have several disjoint routes between each pair of processing nodes, improving the application's resilience to link and node failures. However, these networks complicate the effort to guarantee end-to-end performance, since the system must bound delay at *each link* in a packet's route. This paper presents a router design that supports end-to-end latency and throughput guarantees by scheduling packets at each network link. Although the implementation is geared toward two-dimensional meshes, as shown in Figure 1, the architecture directly extends to other network topologies.

Communication predictability can be improved by assigning priority to time-constrained traffic or to packets that have experienced large delays earlier in their routes [8]. Ultimately, though, bounding worst-case latency requires prior reservation of link and buffer resources, based on the application's anticipated traffic load. Under this traffic contract, the network can provide end-to-end performance guarantees through effective link-scheduling and buffer-allocation policies. The real-time router design handles a wide range of throughput and delay requirements by implementing the *real-time channel* [9] abstraction for packet scheduling, as described in Section 2. A real-time channel is a unidirectional virtual connection between two nodes, with a source traffic specification and end-to-end delay bound; separate parameters for delay and bandwidth permit the model to accommodate a wider range and larger number of connections than other disciplines [10], at the expense of increased implementation complexity.

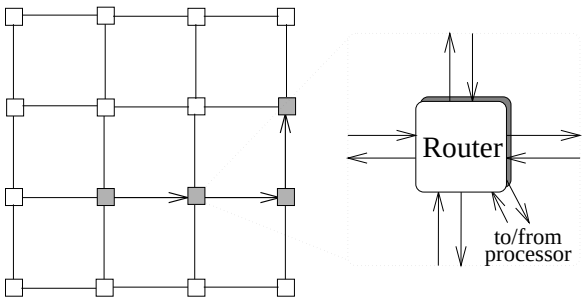


Figure 1: Router in a 4×4 square mesh

At run-time, the network guarantees end-to-end performance through bandwidth regulation and deadline-based packet scheduling at each link. Implementing deadline-based scheduling in software would impose a significant burden on the processing resources at each node and would prove too slow to serve multiple high-speed links. This software would have to sort packets by deadline for each outgoing link, in addition to scheduling and executing application tasks. With high-speed links and tight timing constraints, real-time parallel machines require hardware support for communication scheduling. An efficient, low-cost solution requires a design that integrates this run-time scheduling with packet transmission. Hence, we present a *chip-level* router design that handles run-time packet scheduling, while relegating non-real-time operations (such as admission control and route selection) to the protocol software.

Although deadline-based scheduling bounds worst-case latency for time-constrained traffic, real-time applications also include *best-effort* packets that do not have stringent performance requirements [8–11]; for example, good average delay may suffice for some status and monitoring information. Best-effort traffic should be able to capitalize on the low-latency communication techniques available in modern parallel machines without jeopardizing the performance guarantees of time-constrained packets. Section 3 describes how our design tailors network routing, switching, arbitration, and flow-control policies to the conflicting requirements of these two traffic classes. Time-constrained traffic employs packet switching and small, fixed-size packets to bound worst-case performance, while best-effort packets employ wormhole switching [12] to reduce average latency and minimize buffer space requirements, even for large packets.

Section 4 discusses the router's support for run-time scheduling of time-constrained packets. To reduce hardware complexity, the architecture shares packet buffers and sorting logic between the router's multiple output links. The router overlaps communication scheduling with packet transmission to maximize utilization of the network links. The design limits the complexity of the link scheduler by bounding the range of packet deadlines and handling the effects of clock rollover. Section 5 describes the router implementation, using the Verilog hardware description language and the Epoch silicon compiler. Verilog simulations demonstrate that the design satisfies the performance goals of both traffic classes in a single-chip solution. Section 6 discusses related work, while Section 7 concludes the paper with a discussion of future research directions.

2 Real-Time Channels

Real-time communication requires reservation of bandwidth and buffer resources, coupled with packet scheduling at the network links. The *real-time channel* model [9] provides a useful abstraction for bounding end-to-end network delay, under certain application traffic characteristics.

Traffic parameters: A real-time channel is a unidirectional virtual connection that traverses one or more network links. Since communication is typically periodic, or nearly periodic, in real-time systems, each connection is characterized by its minimum temporal spacing between messages ($I_{\min}$) and maximum message size ($S_{\max}$ bytes). To permit some variation from purely periodic traffic, a connection can generate a burst of up to $B_{\max}$ messages in excess of the periodic restriction $I_{\min}$. Together, these three parameters form a *linear bounded arrival process* [13] that governs a connection's traffic generation at the source node.

End-to-end delay bound: In addition to these traffic parameters, a connection has a bound D on end-to-end message delay, based on the minimum message spacing $I_{\min}$. At the source node, a message m_i generated at time t_i has a *logical arrival time*

$$\ell_0(m_i) = \begin{cases} t_i & \text{if } i = 0 \\ \max\{\ell_0(m_{i-1}) + I_{\min},\ t_i\} & \text{if } i > 0. \end{cases}$$

By basing performance guarantees on these *logical* arrival times, the real-time channels model limits the influence an ill-behaving or malicious connection can have on other traffic in the network. The run-time link scheduler guarantees that message m_i reaches its destination node by its deadline $\ell_0(m_i) + D$.

Per-hop delay bounds: The network does not admit a new connection unless it can reserve sufficient buffer and bandwidth resources without violating the requirements of existing connections [9, 14]. A connection establishment procedure decomposes the connection's end-to-end delay bound D into local delay bounds d_j for each hop in its route such that $d_j \leq I_{\min}$ and $\sum_j d_j \leq D$. Based on the local delay bounds, a message m_i has a logical arrival time

$$\ell_j(m_i) = \ell_{j-1}(m_i) + d_{j-1} \qquad \text{for } j > 0$$

	Traffic	Data Structure
Queue 1	On-time time-constrained packets	Priority queue (by deadline $\ell(m)+d$)
Queue 2	Best-effort packets	First-in-first-out queue
Queue 3	Early time-constrained packets	Priority queue (by logical arrival time $\ell(m)$)

Table 1: Link scheduling queues in real-time channels model

at node j in its route, where $j=0$ corresponds to the source node. Link scheduling ensures that message m_i arrives at node j no later than time $\ell_{j-1}(m_i) + d_{j-1}$, the local deadline at node $j-1$; however, message m_i could reach node j *earlier*, due to variations in delay at previous hops in the route.

Run-time link scheduling: Each link *schedules* time-constrained traffic in order to bound message delay without exceeding the reserved buffer space at intermediate nodes. The scheduler, which employs a multi-class variation of the earliest due-date algorithm [15], gives highest priority to time-constrained messages that have reached their logical arrival time (i.e., $\ell_j(m_i) \leq t$), transmitting the message with the smallest deadline $\ell_j(m_i) + d_j$, as shown in Table 1. If Queue 1 is empty, the link services best-effort traffic from Queue 2, ahead of any early time-constrained messages, thus improving the average performance of best-effort traffic without violating the delay requirements of time-constrained communication. Queue 3 holds early time-constrained traffic, effectively absorbing variations in delay at the previous node; upon reaching its logical arrival time, a message moves from Queue 3 to Queue 1.

Buffer requirements: By postponing the transmission of early time-constrained traffic, the link scheduler avoids overloading the buffer space at the downstream node [9, 10]. If the first two scheduling queues are empty, the link can transmit early time-constrained traffic from Queue 3, as long as these messages are within a small distance $h \geq 0$ of their logical arrival time. Incorporating this *horizon* parameter improves average latency and bandwidth utilization, at the expense of increased buffer requirements at the downstream node. A connection's local delay bound, coupled with the incoming link's horizon parameter, limits the required buffer space at the next node in the route. Node j can receive a message as early as $\ell_j(m_i) - (h_{j-1} + d_{j-1})$, if the incoming link has horizon h_{j-1}; the node can hold a message until its deadline $\ell_j(m_i) + d_j$. If messages arrive as early as possible, and depart as late as possible, then node j could have to store as many as

$$\left\lceil \frac{(h_{j-1} + d_{j-1}) + d_j}{I_{\min}} \right\rceil$$

messages from this connection at the same time. Although each connection could conceivably have its own horizon value, employing a single h parameter allows the link to transmit early traffic directly from the head of Queue 3, without any per-connection data structures.

3 Mixing Traffic Classes

Best-effort and time-constrained traffic have conflicting performance goals that complicate network design. Figure 2 shows the architecture of the real-time router, with separate control and data path for the two traffic classes; solid lines denote the flow of packet data, while dashed lines indicate control information. To insulate the local processor from packet scheduling, the design has separate injection ports for time-constrained and best-effort traffic, while the router coordinates access to a shared reception port and the four outgoing links. Careful selection of router policies, coupled with fine-grain link arbitration, enables time-constrained and best-effort packets to share network bandwidth without sacrificing the performance of either class.

3.1 Switching

To ensure that time-constrained packets meet their delay requirements, the router must have control over bandwidth and memory allocation. In most real-time systems, time-constrained communication consists of 10–20 byte exchanges of command or status information [7]. Consequently, our design restricts time-constrained traffic to small, fixed-size packets, as shown in Table 2; this bounds network access latency and buffering delay while simplifying memory allocation in the router. To ensure predictable consumption of link and buffer resources, time-constrained traffic employs store-and-forward *packet switching*. By buffering packets at each node, packet switching allows each router to independently schedule packet transmissions to satisfy per-hop delay requirements.

However, this approach unduly penalizes the performance of best-effort traffic. Most modern parallel machines employ *cut-through* switching schemes for lower latency and reduced buffer space requirements. In particular, *wormhole switching* permits an arriving packet to proceed directly to the next node in its route, stalling in the network if the outgoing link is not available [12]. In effect, this converts the best-effort scheduling "queue" in Table 1 into a *logical* queue that spans multiple nodes. Instead of storing entire best-effort packets at intermediate nodes, the router simply includes small *flit* (flow control unit) buffers to hold a few

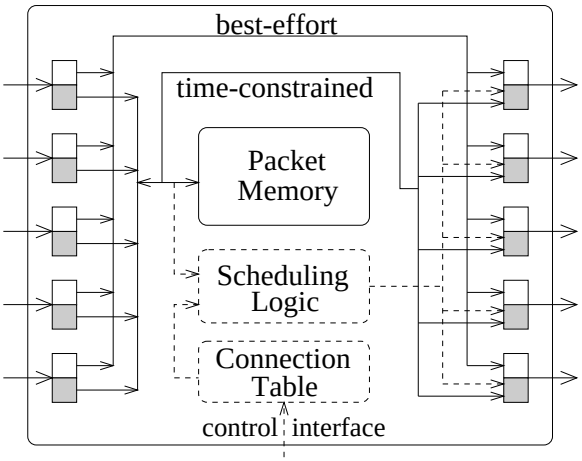


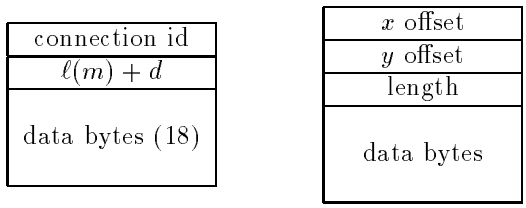
Figure 2: Real-time router architecture

bytes of a packet from each input link; inter-node flow control stalls further transmission of the packet until this buffer space is available. This permits best-effort traffic to use variable-size packets, to reduce or even avoid packetization overheads, without increasing buffer complexity in the router.

3.2 Arbitration

By cutting through intermediate nodes, best-effort packets can avoid unnecessary buffering delay. However, these wormhole packets can stall in the network for an unpredictable amount of time, delaying the advancement of other packets heading for different destinations. The effective mixing of time-constrained and best-effort traffic hinges on controlling the interaction between these two classes [16]. In particular, best-effort packets should not consume arbitrary amounts of bandwidth resources while time-constrained packets await service. To control the interaction between the two traffic classes, the real-time router divides each link into two *virtual channels* [17]. A single bit on each link differentiates between time-constrained and best-effort packets; each link also includes an acknowledgement bit for flow control on the best-effort virtual channel.

Each wormhole virtual channel performs round-robin arbitration on the input links to select an incoming best-effort packet for service, while the packet-switched virtual channel transmits time-constrained packets based on their deadlines. Priority arbitration amongst the virtual channels tightly regulates the intrusion of best-effort traffic on time-constrained packets. This effectively provides *flit-level* preemption of best-effort traffic whenever an on-time time-constrained packet awaits service, while permitting wormhole flits to consume any excess link bandwidth. The link transmits best-effort flits ahead of any *early* time-constrained packets.



(a) Time-constrained (b) Best-effort

Figure 3: Packet formats in real-time router

3.3 Routing

As part of establishing a real-time channel, the network reserves link bandwidth and buffer space along a fixed path between the source and destination nodes; the chosen route depends on the resources available at various nodes and links in the network. Consequently, the real-time router maintains a routing table, indexed on the connection identifier of the arriving time-constrained packet, as shown in Figure 3(a); Section 4 describes how the controlling processor can edit this table as part of a connection establishment protocol. Since a node may wish to send information to a *collection* of destination nodes (i.e., multicast), the router can forward an incoming time-constrained packet to multiple outgoing links; this facilitates efficient, timely communication between a set of cooperating nodes.

In contrast, best-effort traffic does not require resource reservation along packet routes. Instead, the real-time router implements dimension-ordered routing, a shortest-path scheme that completely routes a packet in the x-direction before proceeding in the y-direction to the destination, as shown by the shaded nodes in Figure 1. Dimension-ordered routing avoids packet deadlock in a square mesh [18] and also facilitates an efficient implementation based on x and y offsets in the packet header, as shown in Figure 3(b); the offsets reach zero when the packet has arrived at its destination node. The router could improve best-effort performance by implementing *adaptive* wormhole routing, with additional virtual channels to avoid deadlock, at the expense of increased implementation complexity [19, 20]. In particular, non-minimal adaptive routing would enable best-effort packets to circumvent links with a heavy load of time-constrained traffic.

3.4 Buffer Architecture

The real-time router includes a packet memory for storing time-constrained traffic awaiting access to the outgoing links; in contrast, blocked best-effort packets stall in the network. The router queues time-constrained packets at the output ports to avoid the throughput limitations of input queueing [21]; this permits each output link to select a packet for transmis-

	Time-Constrained	Best-Effort
Switching	Packet switching	Wormhole switching
Packet size	20 bytes	Variable length
Link arbitration	Deadline-driven	Round-robin on input links
Routing	Table-driven multicast	Dimension-ordered unicast
Buffers	Shared output queues	Flit buffers at input links
Flow control	Rate-based	Flit acknowledgements

Table 2: Architectural parameters in real-time router design

sion amongst *all* time-constrained traffic buffered in the router. The reception port and four output links *share* a single packet memory to maximize usage of the available buffer space. To accommodate the aggregate bandwidth of the five input and five output ports, the router stores packets in 10-byte chunks, with demand-driven round-robin arbitration amongst the ports. As shown in Figure 2, each port includes nominal buffer space to avoid stalling the flow of data while waiting for bus access to the packet memory. Similarly, each port includes a small flit buffer to permit continuous transmission of wormhole packets in the absence of link contention.

Similar to many shared-memory switches in high-speed networks [21], our design includes an idle-address FIFO for assigning unused memory locations to arriving time-constrained packets. An incoming packet retrieves an address from this FIFO; upon packet departure, the router returns the location to this idle-address pool. To avoid buffer overflow or packet loss, a real-time channel reserves sufficient buffer slots at each node in its route, as described in Section 2. Although the output ports share a single packet memory, the connection establishment procedure can *logically* partition the memory by limiting the number of packet buffers dedicated to connections on each outgoing link; otherwise, one link could reserve the bulk of the memory slots, limiting the chance of establishing real-time channels on the other outgoing links. By implementing a physically shared memory, the router permits the protocol software to balance the trade-offs between buffer partitioning and complete sharing to enhance future channel admissability.

4 Real-Time Support

Supporting time-constrained communication in a single chip requires careful consideration of the interface to the controlling protocol software. The real-time router permits flexible software control of connection establishment, while implementing efficient run-time packet scheduling on the outgoing ports.

4.1 Control Interface

Establishing a real-time channel requires the application to specify the traffic parameters and performance requirements for the new connection. Admitting a new connection, and selecting a multi-hop route with suitable local delay parameters, is a computationally-intensive procedure [8, 9, 14]. Fortunately, channel establishment typically does not impose tight timing constraints; in most cases, the network can create the required channels before data transfer commences. To permit a single-chip solution, the real-time router relegates these non-real-time operations to the protocol software. Software control also permits greater flexibility in route selection and buffer allocation policies.

Write Command	Fields
Connection parameters	outgoing connection id
	local delay bound d
	bit-mask of output ports
	incoming connection id
Horizon parameter	bit-mask of output ports
	horizon value h

Table 3: Control interface commands

As part of establishing a new real-time channel, each node in the connection's route writes control information into a table in the router. Indexed off the connection identifier, the table stores the channel's local delay bound d and a bit mask for routing incoming packets to the appropriate output port(s); to simplify the design, a multicast connection uses the same value of d for any outgoing ports at the node. To minimize the number of pins on the chip, the controlling processor updates the connection table as a sequence of four write operations, as shown in Table 3. When a time-constrained packet arrives, the router reads the deadline and routing information and assigns a new connection identifier for use at the next node in the packet's route. The router also assigns the packet's local deadline, based on the delay parameter d and the logical arrival time $\ell(m)$.

The packet deadline at one node serves as the logical arrival time at the downstream node in the route. Carrying these logical arrival times in the packet header implicitly assumes that the network routers have a common notion of time, within some bounded clock skew. Although this is not appropriate in a wide-area network context, the tight coupling in parallel machines mini-

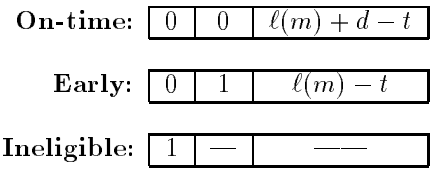


Figure 4: Sorting key for time-constrained packets

mizes the effects of clock skew. Alternatively, the router could store additional information in the connection table to compute $\ell_j(m_i)$ from a packet's actual arrival time and the logical arrival time of the connection's previous packet [22]; however, this approach would require the router to periodically refresh this connection state to correctly handle the effects of clock rollover.

In addition to the connection table, the router maintains a separate horizon parameter h for each outgoing port. As discussed in Section 2, these horizon values permit the router to transmit a time-constrained packet in advance of its logical arrival time, when no on-time packets or best-effort flits await service. The local processor can write the horizon registers through the control interface, as shown in Table 3. Larger horizon values permit earlier transmission of time-constrained packets, but require connections to reserve more buffer space at the downstream node. If necessary, the protocol software could reduce a port's horizon parameter as more connections are established, to free downstream buffer space for reservation by the new connections.

4.2 Scheduling Logic

The real-time router schedules time-constrained traffic for transmission based on logical arrival times and deadlines, as well as the link horizon parameters. To maximize link utilization and channel admissability, the router overlaps run-time communication scheduling with packet transmission on each of the five output ports. As a result, packet size determines the acceptable worst-case scheduling delay, limiting both the maximum number of time-constrained packets and the size of the sorting keys [23]; to facilitate a single-chip solution, our design efficiently handles a moderate number of packets. Since packet sorting can introduce considerable hardware complexity [23–28], particularly when connections have a wide range of delay and bandwidth parameters, the real-time router *shares* the scheduling logic amongst the early and on-time packets headed for any of the five outgoing ports.

Table 1 suggests that each outgoing port requires separate priority queues for early and on-time packets. However, implementing two priority queues for each link would incur significant hardware cost and would require logic to transfer packets from the early queue to the on-time queue; also, multiple packets can reach their logical arrival times *simultaneously*, further complicating movement between the two priority queues. Hence, the real-time router does not attempt to store time-constrained packets in sorted order; instead, the router employs a tree of comparators to select the packet with the smallest key. The base of the tree computes a key for each packet, based on the packet state and the current time t; a bit in the packet key differentiates between early and on-time traffic, as shown in Figure 4.

For on-time traffic, the lower bits of the key represent packet *laxity*, the time remaining till the local deadline expires, whereas the key for early traffic represents the time left before reaching the packet's logical arrival time. Normalizing the packet keys, relative to current time t, allows the rest of the tree to perform simple, unsigned comparison operations, even in the presence of clock rollover. To avoid replicating the scheduling logic, all five outgoing ports share access to a single comparator tree that arbitrates amongst all time-constrained packets, as shown in Figure 5. Pipelining the comparator tree provides the necessary throughput to overlap run-time scheduling with packet transmission on each outgoing port. This also permits the ports to conveniently share the same packet memory. Although this buffer memory stores the actual packet data, the base of the comparator tree maintains a small amount of per-packet state to coordinate run-time scheduling.

As shown in Figure 5, each leaf in the tree stores a logical arrival time $\ell(m)$, a deadline $\ell(m) + d$, and a bit mask of outgoing ports, assigned at packet arrival based on the connection state. The bit mask determines if the leaf is eligible to compete for access to a particular outgoing port. When a port transmits a selected packet, it clears the corresponding field in the leaf's bit mask; a bit mask of zero indicates an empty packet leaf slot and a corresponding idle slot in the packet memory. The base of the tree also determines if packets are early ($\ell(m) > t$) or on-time ($\ell(m) \leq t$) and computes the sorting keys based on the current value of t. At the top of the sorting tree, an additional comparator checks to see if the winning packet is early traffic that falls within the link's horizon parameter; if so, the link transmits this packet, unless best-effort flits await service.

4.3 Handling Clock Rollover

The number of bits in the sorting keys directly affects the latency and implementation complexity of the comparator tree. However, by limiting the size of the keys, the router also restricts the range of local delay bounds d that can be selected by time-constrained connections. To formalize this trade-off, consider a connection traversing consecutive links $j-1$ and j, with local delay parameters d_{j-1} and d_j, respectively, and a horizon parameter h_{j-1} at link $j-1$. A packet can arrive as

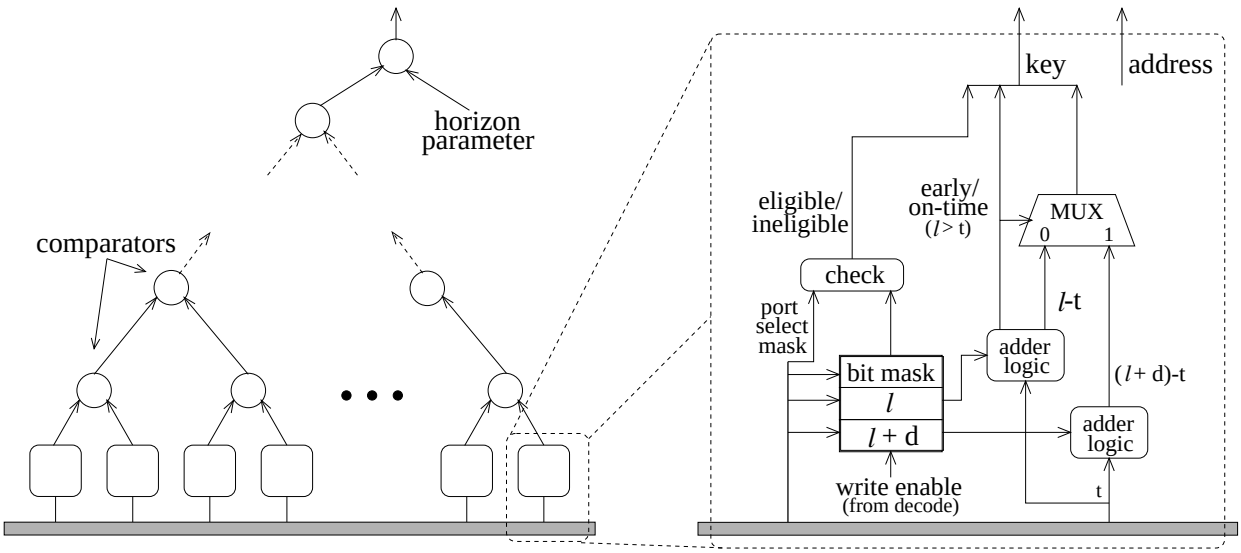


Figure 5: Comparator tree for run-time scheduling

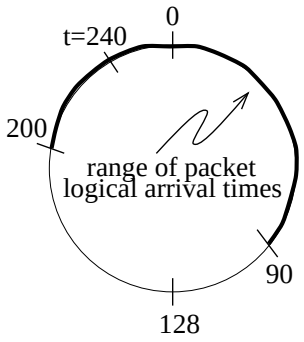


Figure 6: Handling clock rollover with an 8-bit clock

much as $h_{j-1}+d_{j-1}$ time units ahead of its logical arrival time $\ell_j(m)$, if link $j-1$ transmits the packet as early as possible. Similarly, link j must transmit the packet by its deadline $\ell_j(m)+d_j$. Hence, at time t and link j, any packets from this connection have logical arrival times $\ell_j(m) \in [t-d_j,\ t+(h_{j-1}+d_{j-1})]$.

This property permits the router to limit the size of the packet sorting keys, as well as the required number of bits in the on-chip clock, where the clock ticks once per packet transmission time. The router can correctly interpret logical arrival times and deadlines, even in the presence of clock rollover, as long as each connection has $h_{j-1}+d_{j-1}$ and d_j values that are less than *half* the range of the on-chip clock register. For example, Figure 6 shows a range of $\ell_j(m)$ values for different connections under an 8-bit clock, with a range of 256 time units. A packet with $\ell(m)=80$ would be considered early traffic (since $t-80 \geq 128$), while a packet with $\ell(m)=210$ would be considered on-time traffic (since $t-210 < 128$). This enables the leaves of the sorting tree to compute the normalized keys, relative to current time t, using modulo arithmetic.

5 Implementation

5.1 Chip Design

The router chip has been designed using the Verilog hardware description language and the Epoch silicon compiler, with the parameters in Table 4. Using a three-metal, 0.5μm CMOS process, the 123-pin chip has dimensions 8.1 mm $\times$ 8.7 mm for an implementation with 256 time-constrained packets and up to 256 connections. The link-scheduling logic accounts for the majority of the chip area, with the packet memory consuming much of the remaining space. Operating at 50 MHz, the chip can transmit or receive a byte of data on each of its ten ports every 20 nsec; this closely matches the access time of the 10-byte-wide, single-ported SRAM for storing time-constrained traffic. Since time-constrained packets are 20-bytes long, the scheduling logic must select a packet for transmission every 400 nsec for each of the five output ports. To achieve the necessary throughput, the comparator tree consists of a two-stage pipeline, where each stage requires approximately 50 nsec; the boundary between the two pipeline stages consists of a set of latches across a row of comparators.

Although the tree could incorporate up to five pipeline stages, the two-stage design provides sufficient throughput to satisfy the output ports. This suggests that the link scheduler could effectively support a larger number of packets or additional output ports, for a higher-dimensional mesh topology. Alternately, the router design could reduce the hardware cost of the comparator tree by sharing comparator logic between mul-

Parameter	Value
Connections	256
Time-constrained packets	256
Clock (sorting key)	8 (9) bits
Comparator tree pipeline	2 stages
Flit input buffer	10 bytes

(a) Architectural parameters

Parameter	Value
Process	0.5μm 3-metal CMOS
Signal pins	123
Transistors	905,104
Area	8.1 mm $\times$ 8.7 mm
Power	2.3 watts

(b) Chip complexity

Table 4: Router specification

tiple leaves of the tree. In Figure 5, this would combine several leaf units into a single module with a small memory to store the packets' deadlines and logical arrival times; the router could sequence through each module's packets to serialize access to a single comparator at the base of the tree. This would reduce the number of comparators, as well as the loading on the bus to the packet control modules; currently, the design includes a buffer tree to provide the necessary fanout from this bus.

5.2 Experiments

Verilog simulations were used to test a single router chip under a variety of traffic patterns. A preliminary experiment tests the baseline performance of best-effort wormhole packets. To study a multi-hop configuration, the router connects its links in the x and y directions. The packet proceeds from the injection port to the positive x link, then travels from the negative x input link to the positive y direction; after reentering the router on the negative y link, the packet proceeds to the reception port. In this test, a b byte wormhole packet incurs an end-to-end latency of $30 + b$ cycles, where the link transmits one byte in each cycle. This delay is proportional to packet length, with a small overhead for synchronizing the arriving bytes, processing the packet header, and accumulating five-byte chunks for access to the router's internal bus. In contrast, packet switching would introduce additional delay to buffer the packet at each hop in its route.

An additional experiment illustrates how the router schedules time-constrained packets to satisfy delay and throughput guarantees, while allowing best-effort traffic to capitalize on any excess link bandwidth. Figure 7 plots the link bandwidth consumed by best-effort traffic and each of three time-constrained connections with the following parameters, in units of 20-byte slots:

	d	$I_{\min}$
0	8	9
1	5	7
2	3	4

All three connections compete for access to a single network link with horizon parameter $h = 0$, where each

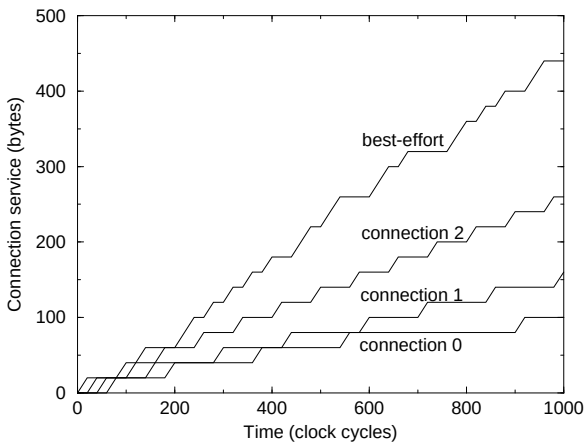


Figure 7: Time-constrained and best-effort service

connection has a continual backlog of traffic. The time-constrained connections receive service in proportion to their throughput requirements, since a packet is not eligible for service till its logical arrival time. Similarly, the link transmits each packet by its deadline, with best-effort flits consuming any remaining link bandwidth.

6 Related Work

This paper complements recent work on support for real-time communication in parallel machines [1–6]. Several projects have proposed mechanisms to improve predictability in the wormhole-switched networks common in modern multicomputers. In the absence of hardware support for priority-based scheduling, application and operating system software can control end-to-end performance by regulating the rate of packet injection at each source node [6]. However, this approach must limit utilization of the communication network to account for possible contention between packets, even from lower-priority traffic. This is a particularly important issue in wormhole networks, since a stalled packet may indirectly block the advancement of other traffic that does not even use the same links. The underlying router architecture can improve predictability by favoring older packets when assigning virtual channels or arbitrating between channels on the same physical link [17].

Although these mechanisms reduce variability in end-to-end latency, more aggressive techniques are necessary to guarantee performance under high network utilization. A router can support multiple classes of traffic, such as user and system packets, by partitioning traffic onto different virtual channels, with priority-based arbitration for access to the network links [17]. Flit-level preemption of low-priority virtual channels can significantly reduce intrusion on the high-priority packets. Still, these coarse-grain priorities do not differentiate between packets with different latency tolerances. With additional virtual channels, the network has greater flexibility in assigning packet priority, perhaps based on the end-to-end delay requirement, and restricting access to virtual channels reserved for higher-priority traffic [3, 4].

Coupled with restrictions on the source injection rate, these policies can bound end-to-end packet latency by limiting the service and blocking times for higher-priority traffic [2]. Although assigning priorities to virtual channels provides some control over packet scheduling, this ties priority resolution to the number of virtual channels. The router can support a finer grain of packet priorities by increasing the number of virtual channels, at the expense of implementation complexity; extra virtual channels incur the cost of additional flit buffers and larger virtual channel identifiers, as well as more complex switching and arbitration logic [20]. Instead of dedicating virtual channels and flit buffers to each priority level, a router can increase priority resolution by adopting a packet-switched design.

The priority-forwarding router chip [5] follows this approach by employing a 32-bit priority field in small, 8-packet priority queues at each input port. The router incorporates a priority-inheritance protocol to limit the effects of priority inversion when a full input buffer limits the transmission of high-priority packets from the previous node; the input buffer's head packet inherits the priority of the highest-priority packet still waiting at the upstream router. In contrast, the real-time router implements a single, shared output buffer that holds up to 256 time-constrained packets, with a link-scheduling and memory reservation model that implicitly avoids buffer overflow. By dynamically assigning an 8-bit packet priority at each node, the real-time router can satisfy a diverse range of end-to-end delay bounds, while permitting best-effort wormhole traffic to capitalize on any excess link bandwidth.

7 Conclusion

Parallel real-time applications impose diverse communication requirements on the underlying interconnection network. The real-time router design supports these emerging applications by bounding packet delay for time-constrained traffic, while ensuring good average performance for best-effort traffic. Low-level control over routing and switching, coupled with fine-grain arbitration at the network links, enables the router to effectively mix these two diverse traffic classes. Careful handling of clock rollover enables the router to support connections with diverse delay and throughput parameters with small packet sorting keys. Sharing sorting logic and packet buffers amongst the five output ports permits a single-chip solution that handles up to 256 time-constrained packets simultaneously.

As future work, we are extending the router architecture to enhance performance and flexibility. In particular, the router can improve link utilization and average latency by using *virtual cut-through switching* [5, 29] for time-constrained traffic; this would permit an arriving packet to proceed directly to its output link if no other packets have smaller sorting keys. We are also considering alternate link-scheduling algorithms that would improve the router's scalability; these algorithms could include approximate versions of real-time channels, as well as new schemes with reduced implementation complexity. This would permit the router to efficiently handle a larger number of time-constrained packets.

To complement the Verilog simulations of the router chip, work is underway to incorporate the real-time router architecture in a multicomputer network simulator [30]. This simulation environment will enable us to evaluate the design under larger network configurations and more diverse traffic patterns, while facilitating direct comparisons to alternate router architectures. These experiments can also explore the ability of the chip to serve as a building block for constructing large, high-speed switches that support the quality-of-service requirements of real-time and multimedia applications.

Acknowledgements

The authors would like to thank Ashish Mehra, Stuart Daniel, and Wu-chang Feng for their comments and suggestions on the paper. Also, the authors appreciate Stuart Daniel's assistance in using Verilog and the Epoch Silicon Compiler. The work reported in this paper was supported in part by the National Science Foundation under grant MIP–9203895. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of NSF.

References

[1] L. R. Welch and K. Toda, "Architectural support for real-time systems: Issues and trade-offs," in *Proc. International Workshop on Real-Time Computing Systems and Applications*, December 1994.

[2] M. W. Mutka, "Using rate monotonic scheduling technology for real-time communications in a wormhole net-

work," in *Proc. Workshop on Parallel and Distributed Real-Time Systems*, April 1994.

[3] J.-P. Li and M. W. Mutka, "Priority based real-time communication for large scale wormhole networks," in *Proc. International Parallel Processing Symposium*, pp. 433–438, April 1994.

[4] A. Saha, "Simulator for real-time parallel processing architectures," in *Proc. IEEE Annual Simulation Symposium*, pp. 74–83, April 1995.

[5] K. Toda, K. Nishida, E. Takahashi, N. Michell, and Y. Yamaguchi, "Design and implementation of a priority forwarding router chip for real-time interconnection networks," *International Journal of Mini and Microcomputers*, vol. 17, no. 1, pp. 42–51, 1995.

[6] R. Games, A. Kanevsky, P. Krupp, and L. Monk, "Real-time communications scheduling for massively parallel processors," in *Proc. Real-Time Technology and Applications Symposium*, pp. 76–85, May 1995.

[7] R. S. Raji, "Smart networks for control," *IEEE Spectrum*, vol. 31, pp. 49–55, June 1994.

[8] C. M. Aras, J. F. Kurose, D. S. Reeves, and H. Schulzrinne, "Real-time communication in packet-switched networks," *Proceedings of the IEEE*, vol. 82, pp. 122–139, January 1994.

[9] D. D. Kandlur, K. G. Shin, and D. Ferrari, "Real-time communication in multi-hop networks," *IEEE Trans. on Parallel and Distributed Systems*, vol. 5, pp. 1044–1056, October 1994.

[10] H. Zhang, "Providing end-to-end performance guarantees using non-work-conserving disciplines," *Computer Communications*, vol. 18, pp. 769–781, October 1995.

[11] Y. Ofek and M. Yung, "The integrated MetaNet architecture: A switch-based multimedia LAN for parallel computing and real-time traffic," in *Proc. IEEE INFOCOM*, pp. 802–811, 1994.

[12] W. J. Dally and C. L. Seitz, "The torus routing chip," *Journal of Distributed Computing*, vol. 1, no. 3, pp. 187–196, 1986.

[13] R. L. Cruz, "A calculus for network delay, part I: Network elements in isolation," *IEEE Trans. Information Theory*, vol. 37, pp. 114–131, January 1991.

[14] Q. Zheng and K. G. Shin, "On the ability of establishing real-time channels in point-to-point packet-switched networks," *IEEE Trans. Communications*, pp. 1096–1105, February/March/April 1994.

[15] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of the ACM*, vol. 20, pp. 46–61, January 1973.

[16] J. Rexford and K. G. Shin, "Support for multiple classes of traffic in multicomputer routers," in *Proc. Parallel Computer Routing and Communication Workshop*, pp. 116–130, May 1994.

[17] W. Dally, "Virtual-channel flow control," *IEEE Trans. Parallel and Distributed Systems*, vol. 3, pp. 194–205, March 1992.

[18] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. Computers*, vol. C-36, no. 5, pp. 547–553, May 1987.

[19] L. Ni and P. McKinley, "A survey of wormhole routing techniques in direct networks," *IEEE Computer*, pp. 62–76, February 1993.

[20] K. Aoyama and A. Chien, "Cost of adaptivity and virtual lanes in a wormhole router," *Journal of VLSI Design*, vol. 2, no. 4, pp. 315–333, 1995.

[21] F. A. Tobagi, "Fast packet switch architectures for broadband integrated services digital networks," *Proceedings of the IEEE*, vol. 78, pp. 133–167, January 1990.

[22] Q. Zheng, K. G. Shin, and C. Chen, "Real-time communication in ATM," in *Proc. Annual Conference on Local Computer Networks*, pp. 156–164, October 1994.

[23] D. Picker and R. D. Fellman, "Scaling and performance of a priority packet queue for real-time applications," in *Proc. Real-Time Systems Symposium*, pp. 56–62, December 1994.

[24] H. J. Chao, "A novel architecture for queue management in the ATM network," *IEEE Journal on Selected Areas in Communications*, vol. 9, pp. 1110–1118, September 1991.

[25] P. E. Boyer, F. M. Guillemin, M. J. Servel, and J.-P. Coudreuse, "Spacing cells protects and enhances utilization of ATM network links," *IEEE Network Magazine*, pp. 38–49, September 1992.

[26] E. Wallmeier and T. Worster, "The Spacing Policer, an algorithm for efficient peak bit rate control in ATM networks," in *Proc. International Switching Symposium*, pp. 22–26, October 1992.

[27] J. Liebeherr and D. Wrege, "Versatile packet multiplexer for quality-of-service networks," in *Proc. IEEE International Symposium on High Performance Distributed Computing*, pp. 148–155, August 1995.

[28] J. Rexford, A. Greenberg, and F. Bonomi, "Hardware-efficient fair queueing architectures for high-speed networks," in *Proc. IEEE INFOCOM*, March 1996.

[29] P. Kermani and L. Kleinrock, "Virtual cut-through: A new computer communication switching technique," *Computer Networks*, vol. 3, pp. 267–286, September 1979.

[30] J. Rexford, J. Dolter, W. Feng, and K. G. Shin, "PP-MESS-SIM: A simulator for evaluating multicomputer interconnection networks," in *Proc. IEEE Annual Simulation Symposium*, pp. 84–93, April 1995.