A Router Architecture for Real-Time Communication in Multicomputer Networks

Jennifer Rexford, John Hall, and Kang G. Shin

Abstract— **Parallel machines have the potential to satisfy the large computational demands of real-time applications. These applications require a predictable communication network, where *time-constrained* traffic requires bounds on throughput and latency while good average performance suffices for *best-effort* packets. This paper presents a new router architecture that tailors low-level routing, switching, arbitration, flow-control, and deadlock-avoidance policies to the conflicting demands of each traffic class. The router implements bandwidth regulation and deadline-based scheduling, with packet switching and table-driven multicast routing, to bound end-to-end delay and buffer requirements 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 includes a novel packet scheduler that shares link-scheduling logic across the multiple output ports, while masking the effects of clock rollover on the represention of packet eligibility times and deadlines. Using the Verilog hardware description language and the Epoch silicon compiler, we demonstrate that the router design meets the performance goals of both traffic classes in a single-chip solution. Verilog simulation experiments on a detailed timing model of the chip show how the implementation and performance properties of the packet scheduler scale over a range of architectural parameters.**

Keywords: **Multicomputer router, real-time communication, link scheduling, wormhole switching, packet switching**

I. Introduction

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 the large computational demands. Real-time parallel computing hinges on effective policies for placing and scheduling communicating tasks in the system to ensure that critical operations complete by their deadlines. Ultimately, a parallel or distributed real-time system relies on an interconnection network that can provide throughput and delay guarantees for critical communication between cooperating tasks; this communication may have diverse performance requirements, depending on the application [1]. However, instead of guaranteeing bounds on *worst-case* communication latency, most existing multicomputer network designs focus on providing good *average* network throughput and packet delay. Consequently, recent years have seen increasing interest in developing interconnection networks that provide performance guarantees in parallel machines [2–8].

Real-time systems employ a variety of network architectures, depending on the application domain and the performance requirements. Although prioritized bus and ring networks are commonly used in small-scale real-time systems [9], 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. To deliver predictable communication performance in multi-hop networks, we present a novel router architecture that supports end-to-end delay and throughput guarantees by scheduling packets at each network link. Our prototype implementation is geared toward two-dimensional meshes, as shown in Figure 1; such topologies have been widely used as the interconnection network for a variety of commercial parallel machines. The design directly extends to a broad set of topologies, including the class of k-ary n-cube networks; with some changes in the routing of best-effort traffic, the proposed architecture applies to arbitrary point-to-point 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 [10]. Ultimately, though, bounding worst-case communication 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. To handle a wide range of bandwidth and delay requirements, the real-time router implements the *real-time channel* [11–13] abstraction for packet scheduling, as described in Section II. Conceptually, a real-time channel is a unidirectional virtual connection between two processing nodes, with a source traffic specification and an end-to-end delay bound. Separate parameters for bandwidth and delay permit the model to accommodate a wider range and larger number of connections than other service disciplines [14–16], at the expense of increased implementation complexity.

The real-time channel model guarantees end-to-end performance through a combination of bandwidth regulation and deadline-based scheduling at each link. Implementing packet scheduling in software would impose a significant burden on the processing resources at each node and

The work reported in this paper was supported in part by the National Science Foundation under grant MIP–9203895 and the Office of Naval Research under grants N00014–94–1–0229. 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 or ONR.

J. Rexford is with AT&T Labs – Research in Florham Park, New Jersey, and J. Hall and K. G. Shin are with the University of Michigan in Ann Arbor, Michigan.

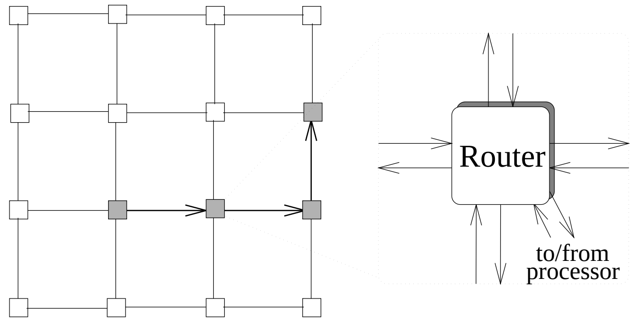


Fig. 1. **Router in a Mesh Network:** This figure shows a router in a 4×4 square mesh of processing nodes. To communicate with another node, a processor injects a packet into its router; then, the packet traverses one or more links before reaching the reception port of the router at the destination node.

would prove too slow to serve multiple high-speed links. This software would have to rank 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 bandwidth regulation and deadline-based scheduling, while relegating non-real-time operations (such as admission control and route selection) to the network protocol software.

Although deadline-based scheduling bounds the worst-case latency for time-constrained traffic, real-time applications also include *best-effort* packets that do not have stringent performance requirements [10, 11, 15, 17]; for example, good average delay may suffice for some status and monitoring information, as well as the protocol for establishing real-time channels. 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 III describes how our design tailors network routing, switching, arbitration, flow-control, and deadlock-avoidance policies to the conflicting performance requirements of these two traffic classes. Time-constrained traffic employs packet switching and small, fixed-sized packets to bound worst-case performance, while best-effort packets employ wormhole switching [18] to reduce average latency and minimize buffer space requirements, even for large packets. The router implements deadlock-free, dimension-ordered routing for best-effort packets, while permitting the protocol software to select arbitrary multicast routes for the time-constrained traffic; together, flexible routing and multicast packet forwarding provide efficient group communication between cooperating real-time tasks.

Section IV describes how the network can reserve buffer and link resources in establishing time-constrained connections. In addition to managing the packet memory and connection data structures, the real-time router effectively handles the effects of clock rollover in computing scheduling keys for each packet. The router overlaps communication scheduling with packet transmission to maximize utilization of the network links. To reduce hardware complexity, the architecture shares packet buffers and sorting logic amongst the router's multiple output links, as discussed in Section V; a hybrid of serial and parallel comparison operations enables the scheduler to trade space for time to further reduce implementation complexity. Section VI describes the router implementation, using the Verilog hardware description language and the Epoch silicon compiler. The Epoch implementation demonstrates that the router can satisfy the performance goals of both traffic classes in an affordable, single-chip solution. Verilog simulation experiments on a detailed timing model of the chip show the correctness of the design and investigate the scaling properties of the packet scheduler across a range of architectural parameters. Section VII discusses related work on real-time multicomputer networks, while Section VIII concludes the paper with a summary of the research contributions and future directions.

II. Real-Time Channels

Real-time communication requires advance reservation of bandwidth and buffer resources, coupled with run-time scheduling at the network links. The *real-time channel* model [11] 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. In most real-time systems, application tasks exchange messages on a periodic, or nearly periodic, basis. As a result, the real-time channel model characterizes each connection by its minimum spacing between messages ($I_{\min}$ time units) and maximum message size ($S_{\max}$ bytes), resulting in a maximum transfer rate of $S_{\max}/I_{\min}$ bytes per unit time. 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* [19] 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 channel 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

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

TABLE I

Real-Time Channel Scheduling Model: Under the real-time channel model, each link transmits traffic from three scheduling queues. To provide delay guarantees to time-constrained connections, the link gives priority to the *on-time* time-constrained messages in Queue 1 over the *best-effort* traffic in Queue 2. Queue 3 serves as a staging area for holding any *early* time-constrained messages.

existing connections [11, 20]. 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 $d_j \le I_{\min}$ and $\sum_j d_j \le D$. Based on the local delay bounds, a message m_i has a logical arrival time

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

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 $j-1$. In fact, message m_i may reach node j *earlier*, due to variations in delay at previous hops in the route. The scheduler at node j ensures that such "early" arrivals do not interfere with the transmission of "on-time" messages from other connections.

Run-time link scheduling: Each link schedules time-constrained traffic, based on logical arrival times and deadlines, 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 [21], gives highest priority to time-constrained messages that have reached their logical arrival time (i.e., $\ell_j(m_i) \le t$), transmitting the message with the smallest deadline $\ell_j(m_i) + d_j$, as shown in Table I. If Queue 1 is empty, the link services best-effort traffic from Queue 2, ahead of any early time-constrained messages (i.e., $\ell_j(m_i) > t$). This improves 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.

Link horizon parameter: By delaying the transmission of early time-constrained messages, the link scheduler can avoid overloading the buffer space at the downstream node [11, 15, 16]. Still, the scheduler could potentially improve link utilization and average latency by transmitting early messages from Queue 3 when the other two scheduling queues are empty. To balance this trade-off between buffer requirements and average performance, the link can transmit an early time-constrained message from Queue 3, as long as the message is within a small horizon $h \ge 0$ of its logical arrival time (i.e., $\ell_j(m_i) \le t + h$). Larger values of h permit the link to transmit more early time-constrained traffic, at the expense of increased memory requirements at the downstream node. Although each connection could conceivably have its own h value, employing a single horizon parameter allows the link to transmit early traffic directly from the head of Queue 3, without any per-connection data structures.

Buffer requirements: To avoid buffer overflow or message loss, a connection must reserve sufficient memory for storing traffic at each node in its route. The required buffer space at node j depends on the connection's local delay bound d_j, as well as the horizon parameter h_{j-1} for the incoming link. In particular, node j can receive a message from node $j-1$ as early as $\ell_j(m_i) - (d_{j-1} + h_{j-1})$, if node $j-1$ transmits the message at the earliest possible time. In the worst case, node j can hold a message until its deadline $\ell_j(m_i) + d_j$. Hence, for this connection, $\ell_j(m_i) \in [t - d_j, t + d_{j-1} + h_{j-1}]$ for any messages m_i stored at node j at time t. If a connection has messages arrive as early as possible, and depart as late as possible, then node j could have to store as many as

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

messages from this connection at the same time. By reserving buffer and bandwidth resources in advance, the real-time channel model guarantees that every message arrives at its destination node by its deadline, independent of other best-effort and time-constrained traffic in the network.

III. Mixing Best-Effort and Time-Constrained Traffic

Although the real-time channel model bounds the worst-case performance of time-constrained messages, the scheduling model in Table I can impose undue restrictions on the packet size and flow-control schemes for best-effort traffic. To overcome these limitations, we propose a router architecture that tailors its low-level communication policies to the unique demands of the two traffic classes. Fine-grain, priority-based arbitration at the network links permits the best-effort traffic to capitalize on the low-latency techniques in modern multicomputer networks without sacrificing the performance guarantees of the time-constrained connections. Figure 2 shows the high-level architecture of the real-time router, with separate control and data path for the two traffic classes.

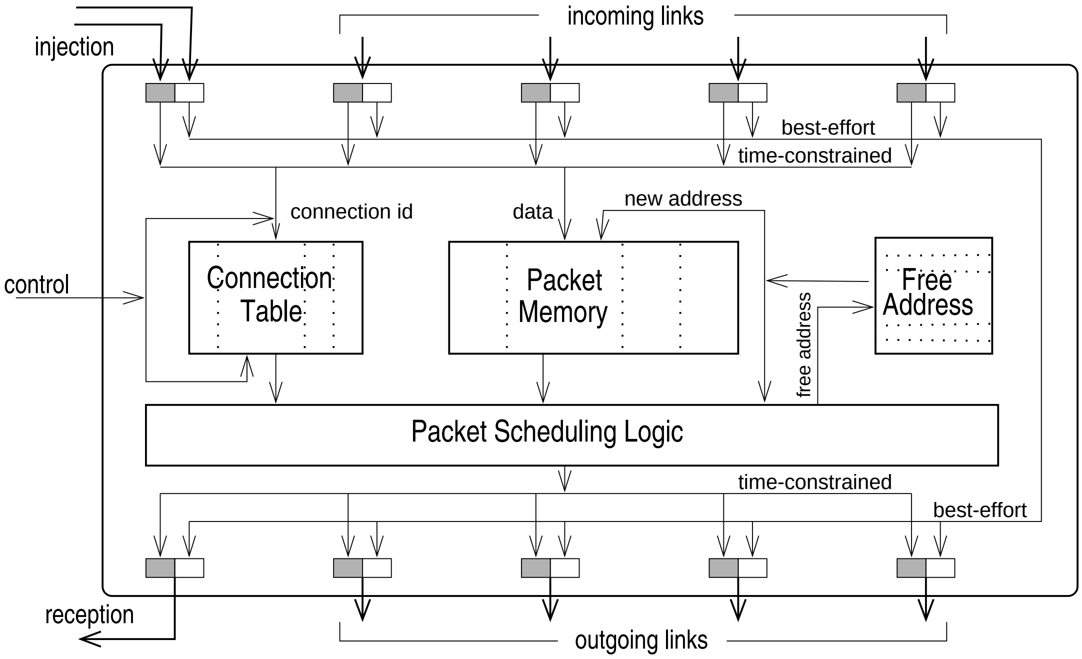


Fig. 2. **Real-Time Router:** This figure shows the real-time router architecture, with separate control and data path for best-effort and time-constrained packets. The router includes a packet memory, connection routing table, and scheduling logic to support delay and bandwidth guarantees for time-constrained traffic. To connect to the local processor, the router exports a control interface, a reception port, and separate injection ports for each traffic class.

A. Complementary Switching Schemes

To ensure that time-constrained connections meet their delay requirements, the router must have control over bandwidth and memory allocation. For example, suppose that a time-constrained message arrives with a tight deadline (i.e., $\ell(m_i) + d - t$ is small), while the outgoing link is busy transmitting other traffic. To satisfy this tight timing requirement, the outgoing link must stop servicing any lower-priority messages within a small, bounded amount of time. This introduces a direct relationship between connection admissibility and the maximum packet size of the time-constrained and best-effort traffic sharing the link. In most real-time systems, time-constrained communication consists of 10–20 byte exchanges of command or status information [9]. Consequently, the real-time router restricts time-constrained traffic to small, fixed-size packets that can support a distributed memory read or write operation. This bounds link 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. To improve average performance, the time-constrained traffic could conceivably employ *virtual cut-through* switching [22] to allow an incoming packet to proceed directly to an idle outgoing link. However, in contrast to traditional virtual cut-through switching of best effort traffic, the real-time router cannot forward a time-constrained packet without first assessing its logical arrival time (to ensure that the downstream router has sufficient buffer space for the packet) and computing the packet deadline (which serves as the logical arrival time at the downstream router). To avoid this extra complexity and overhead, the initial design of the real-time router implements store-and-forward packet switching, which has the same worst-case performance guarantees as virtual cut-through switching. A future implementation could employ virtual cut-through switching to reduce the average latency of the time-constrained traffic.

Although packet switching delivers good, predictable performance to small, time-constrained packets, this approach would significantly degrade the average latency of long, best-effort packets. Even in a lightly-loaded network, end-to-end latency under packet switching is proportional to the product of packet size and the length of the route. Instead, the best-effort traffic can employ *wormhole switching* [18] for lower latency and reduced buffer space requirements. Similar to virtual cut-through switching, wormhole switching permits an arriving packet to proceed directly to the next node in its route. However, when the outgoing link is not available, the packet stalls *in the network* instead of buffering entirely within the router.

In effect, wormhole switching converts the best-effort scheduling "queue" in Table I into a *logical* queue that spans multiple nodes. The router simply includes small five-byte *flit* (flow control unit) buffers [23] to hold a few bytes of a packet from each input link. When an incoming packet fills these buffers, inter-node flow control halts further transmission from the previous node until more space is available; once the five-byte chunk proceeds to a buffer at the outgoing link, the router transmits an acknowledg-

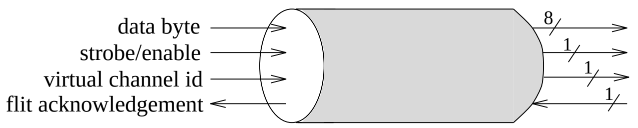
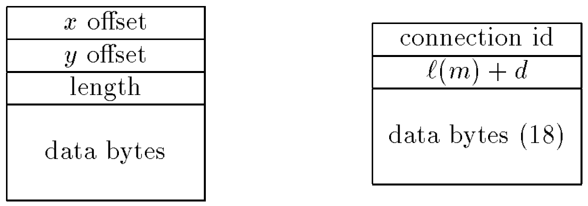


Fig. 3. **Link Encoding:** In the real-time router, each link can transmit a byte of data, along with a strobe signal and a virtual channel identifier. In the reverse direction, an acknowledgment bit indicates that the router can store another flit on the best-effort virtual channel.



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

Fig. 4. **Packet Formats:** This figure illustrates the packet formats for best-effort and time-constrained packets in the real-time router. Best-effort packets consist of a two-byte routing header and a one-byte length field, along with the variable-length data. Time-constrained packets are 20 bytes long and include the connection identifier and the deadline from the previous hop in the route, which serves as the logical arrival time at the current router.

ment bit to signal the upstream router to start sending the next flit. This fine-grain, per-hop flow control permits best-effort traffic to use large variable-sized packets, reducing or even avoiding packetization overheads, without increasing buffer complexity in the router. The combination of wormhole and packet switching, with best-effort traffic consuming small flit buffers and time-constrained connections reserving packet buffers, results in an effective partitioning of router resources.

B. Separate Logical Resources

Even though wormhole and packet switching exercise complementary buffer resources, best-effort and time-constrained traffic still share access to the same network links. To provide tight delay guarantees for time-constrained connections, the router must bound the time that the variable-sized, wormhole packets can stall the forward progress of on-time, time-constrained traffic. However, a blocked wormhole packet can hold link resources at a chain of consecutive routers in the network, indirectly delaying the advancement of other traffic that does not even use the same links. This complicates the effort to provision the network to bound worst-case end-to-end latency, as discussed in the treatment of related work in Section VII. In order to control the interaction between the two traffic classes, the real-time router divides each link into two *virtual channels* [23]. A single bit on each link differentiates between time-constrained and best-effort packets, as shown in Figure 3; each link also includes an acknowledgment 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 and logical arrival times. Priority arbitration between the two virtual channels tightly regulates the intrusion of best-effort traffic on time-constrained packets on each outgoing link. 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. In a separate simulation study, we have demonstrated the effectiveness of using flit-level priority arbitration policies to mix best-effort wormhole traffic and time-constrained packet-switched traffic [24–26].

While the real-time router gives preferential treatment to time-constrained traffic, the outgoing links transmit best-effort flits ahead of any *early* time-constrained packets, consistent with the policies in Table I. Although this arbitration mechanism ensures effective scheduling of the traffic on the outgoing links and the reception port, the best-effort and time-constrained packets could still contend for resources at the *injection* port at the source node. The local processor could solve this problem by negotiating between best-effort and time-constrained traffic at the injection port, but this would require the processor to perform flit-level arbitration. Instead, the real-time router includes a dedicated injection port for each traffic class. The two injection ports, coupled with the low-level arbitration on the outgoing links, ensure that time-constrained traffic has fine-grain preemption over the best-effort packets across the entire path through the network, while allowing best-effort packets to capitalize on any remaining link bandwidth.

C. Buffering and Packet Forwarding

To support the multiple incoming and outgoing ports, the real-time router design requires high throughput for receiving, storing, and transmitting packets. Internally, the router isolates the best-effort and time-constrained traffic on separate buses to increase the throughput and reduce the complexity of the arbitration logic. Each incoming and outgoing port includes nominal buffer space to avoid stalling the flow of data while waiting for access to the bus. The best-effort bus is one flit wide and performs round-robin arbitration among the flit buffers at the incoming ports. Running at the same speed as the byte-wide input ports, this five-byte bus has sufficient throughput to accommodate a peak load of best-effort traffic. Transferring best-effort packets in five-byte chunks incurs a small initial transmission delay at each router, which could be reduced by using a crossbar switch; however, we employ a shared bus for the sake of simplicity. Other recent multicomputer router architectures have used a wide bus for flit transfer [27, 28].

The structure and placement of packet buffers plays a large role in the router's ability to accommodate the performance requirements of time-constrained connections. The simplest solution places a separate queue at each input link. However, input queuing has throughput limitations [29], since a packet may have to wait behind other traffic destined for a different outgoing link. In addition, queuing packets at the incoming links complicates the effort to schedule outgoing traffic based on delay and throughput requirements. Instead, the real-time router queues time-constrained packets at the output ports; the router shares a single packet memory among the multiple output ports to maximize the network's ability to accommodate time-constrained connections with diverse buffer requirements. To accommodate the aggregate memory 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.

Since time-constrained traffic is not served in a first-in first-out order, the real-time router must have a data structure that records the idle memory locations in the packet buffer. Similar to many shared-memory switches in high-speed networks, the real-time router maintains an idle-address pool [29], implemented as a stack. This stack consists of a small memory, which stores the address of each free location in the packet buffer, and a pointer to the first entry. Initially, the stack includes the address of each location in the packet memory. An incoming packet retrieves an address from the top of the stack and increments the stack pointer to point to the next available entry. Upon packet departure, the router decrements this pointer and returns the free location to the top of stack. The idle-address stack always has at least one free address when a new packet arrives, since the real-time channel model never permits the time-constrained traffic to overallocate the buffer resources.

D. Routing and Deadlock-Avoidance

Although wormhole switching reduces the buffer requirements and average latency for best-effort traffic, the low-level inter-node flow control could potentially introduce cyclic dependencies between stalled best-effort packets. To avoid these cycles, 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 [30] and also facilitates an efficient implementation based on x and y offsets in the packet header, as shown in Figure 4(a); the offsets reach zero when the packet has arrived at its destination node. To improve the performance of best-effort traffic, an enhanced version of the router could support *adaptive* wormhole routing and additional virtual channels, at the expense of increased implementation complexity [31,32]. In particular, *non-minimal* adaptive routing would enable best-effort packets to circumvent links with a heavy load of time-constrained traffic.

Although routing is closely tied with deadlock-avoidance for best-effort packets, the real-time router need not dictate a particular routing scheme for the time-constrained traffic. Instead, each time-constrained connection has a fixed path through the network, based on a table in each router; this table is indexed by the connection identifier field in the header of each time-constrained packet, as shown in Figure 4(b). As part of establishing a real-time channel, the network protocol software can select a fixed path from the source to the destination(s), based on the available bandwidth and buffer resources at the routers. The protocol software can employ a variety of algorithms for selecting unicast and multicast routes based on the resources available in the network [33]. Once the connection establishment protocol reserves buffer and bandwidth resources for a real-time channel, the combination of bandwidth regulation and packet scheduling prevents packet deadlock for time-constrained traffic. Table II summarizes how the real-time router employs these and other policies to accommodate the conflicting performance requirements of the two traffic classes.

IV. Managing Time-Constrained Connections

A real-time multicomputer network must have effective mechanisms for establishing connections and scheduling packets, based on the delay and throughput requirements of the time-constrained traffic. To permit a single-chip implementation, the real-time router offloads non-real-time operations, such as route selection and admission control, to the network protocol software. At run-time, the router coordinates access to buffer and link resources by managing the packet memory and the connection data structures. In addition, the router architecture introduces efficient techniques for bounding the range of logical arrival times and deadlines, to limit scheduler delay and implementation complexity.

A. Route Selection and Admission Control

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 [10,11,20]. Fortunately, channel establishment typically does not impose tight timing constraints, in contrast to the actual data transfer which requires explicit guarantees on minimum throughput and worst-case delay. In fact, in most cases, the network can establish the required time-constrained connections before the application commences. To permit a single-chip solution, the real-time router relegates these non-real-time operations to the protocol software. The network could select routes and admit new connections through a centralized server or a distributed protocol. In either case, this protocol software can use the best-effort virtual network, or even a set of dedicated time-constrained connections, to exchange information to select a route and provision resources for each new connection.

The route selected for a connection depends on the traffic characteristics and performance requirements, as well as the available buffer and bandwidth resources in the network. As part of establishing a new real-time channel, the protocol software assigns a unique connection identifier at each hop in the route. Then, each node in the route writes

	Time-Constrained	Best-Effort
Switching	Packet switching	Wormhole switching
Packet size	Small, fixed size	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 acknowledgments

TABLE II

Architectural Parameters: This table summarizes how the real-time router supports the conflicting performance requirements of time-constrained and best-effort traffic.

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 III

Control Interface Commands: This table summarizes the control commands used to configure the real-time router.

control information into the router's connection table, as shown in Table III. At run-time, this table is indexed by the connection identifier field of each incoming time-constrained packet, as shown in Figure 4(b). To minimize the number of pins on the router chip, the controlling processor updates this table as a sequence of four, one-byte operations that specify the incoming connection identifier and the three fields in the table. After closing a connection, the network protocol software can reuse the connection identifier by overwriting the entry in the routing table. The processor uses the same control interface to set the horizon parameters h for each of the five outgoing ports.

As shown in Table III, the routing table stores the connection's identifier at the next node, the local delay bound d, and a bit mask for directing traffic to the appropriate outgoing port(s). When a packet arrives, the router indexes the table with the incoming connection identifier and replaces the header field with the new identifier for the downstream router. At the same time, the router computes the packet's deadline from the logical arrival time in the packet header and the local delay bound in the connection table. Finally, the bit mask permits the router to forward an incoming packet to multiple outgoing ports, allowing the network protocol software to establish *multicast* real-time channels. This facilitates efficient, timely communication between a set of cooperating nodes. To simplify the design, the real-time router requires a multicast connection to use the same value of d for each of its outgoing ports at a single node. Then, based on the bit mask in the routing table, the router queues the updated packet for transmission on the appropriate outgoing port(s).

By implementing a shared packet memory, the real-time router can store a single copy of each multicast packet, removing the packet only after it has been transmitted by each output port selected in the bit mask. The shared packet memory also permits the network protocol software to employ a wide variety of buffer allocation policies. On the one extreme, the route selection and admission control protocols could allocate packet buffers to any new connection, independent of its outgoing link. However, this could allow a single link to consume the bulk of the memory locations, reducing the chance of establishing time-constrained connections on the other outgoing links. Instead, the admission control protocol should bound the amount of buffer space available to each of the five outgoing ports. Similarly, the network could limit the size of the link horizon parameters h to reduce the amount of memory required by each connection. In particular, at run-time, a higher-level protocol could reduce the h values of a router's incoming links when the node does not have sufficient buffer space to admit new connections.

B. Handling a Clock with Finite Range

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, as shown in Figure 4(b), 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 minimizes 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 [34]; however, this approach would require the router to periodically refresh this connection state to correctly handle the effects of clock rollover. Instead, the real-time router avoids this overhead by capitalizing on the tight coupling between nodes to assume synchronized clocks.

Even with synchronized clocks, the real-time router cannot completely ignore the effects of clock rollover. To schedule time-constrained traffic, the router architecture includes a real-time clock, implemented as a counter that increments once per packet transmission time. For a practical implementation, the router must limit the number of bits b used to represent the logical arrival times and deadlines of time-constrained packets. Since logical arrival times continually increase, the design must use modulo arithmetic to compute packet deadlines and schedule traffic for transmission. As a result, the network must restrict the logical arrival times that can exist in a router at the same time; otherwise, the router cannot correctly distinguish between different packets awaiting access to the outgoing link.

Selecting a value for b introduces a fundamental trade-off between connection admissibility and scheduler complexity. To select a packet for transmission, the scheduler must compare the deadlines and logical arrival times of the time-constrained packets; for example, the data structures in Table I require comparison operations to enqueue/dequeue packets. Larger values of b would increase the hardware cost and latency for performing these packet comparison operations. However, smaller values of b would restrict the network's ability to select large delay bounds d and horizon parameters h for time-constrained connections. The network protocol software can limit the delay and horizon parameters, based on the value of b imposed by the router implementation. Alternatively, in implementing the router, a designer could select a value for b based on typical requirements for the expected real-time applications.

To formalize the trade-off between complexity and admissibility, consider a connection traversing consecutive links $j-1$ and j, with local delay parameters d_{j-1} and d_j, respectively, where link $j-1$ has horizon parameter h_{j-1}. As discussed in Section II, a packet can arrive as much as $h_{j-1}+d_{j-1}$ time units ahead of its logical arrival time $\ell_j(m)$ and depart as late as its deadline $\ell_j(m)+d_j$. Consequently,

$$\ell_j(m_i) \in [t - d_j, t + d_{j-1} + h_{j-1}]$$

for any messages m_i from this connection at time t. The network must ensure that the router can differentiate between the full range of logical arrival times in this set. The router can correctly interpret logical arrival times and deadlines, even in the presence of clock rollover, as long as every connection has $h_{j-1}+d_{j-1}$ and d_j values that are less than *half* the range of the on-chip clock. That is, the router requires $d_j < 2^{b-1}$ and $d_{j-1}+h_{j-1} < 2^{b-1}$ for all connections sharing the link.

Under this restriction, the router can compare packets based on their logical arrival times and deadlines by using modulo arithmetic. For example, suppose $b = 8$ (i.e., the clock has a range of 256 time units) and the connections all satisfy $d_j \leq 40$ and $d_{j-1}+h_{j-1} \leq 106$. At time $t=240$, this configuration corresponds to Figure 5. Any early packets have logical arrival times between 240 and 346, modulo 256. For example, a packet with $\ell(m)=80$ would be considered early traffic (since $(80-t) \bmod 256 = 96 < 128$). Similarly, any on-time packets have logical arrival times between 200 and 240. For example, a packet with $\ell(m) = 210$ would be considered on-time traffic (since $(t - 210) \bmod 256 = 30 < 128$). Since on-time packets have $\ell_j(m_i) \leq t$, their deadlines satisfy $\ell_j(m_i) + d_j \in [t, t + d_j]$. Hence, these deadlines also fall within the necessary range in Figure 5, allowing the router to compute $(\ell_j(m_i)+d_j-t) \bmod 256$ to compare on-time packets based on their deadlines.

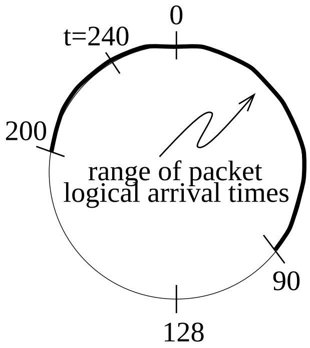


Fig. 5. **Handling Clock Rollover:** This figure illustrates the effects of clock rollover with an 8-bit clock, where the current time is $t=240 \pmod{256}$. In the example, all connections satisfy $d_j \leq 40$ and $d_{j-1}+h_{j-1} \leq 106$, ensuring that the router can correctly compare $\ell_j(m_i)$ to t to distinguish between on-time and early packets.

V. Scheduling Time-Constrained Packets

To satisfy connection delay, throughput, and buffer requirements, each outgoing port must schedule time-constrained packets based on their logical arrival times and deadlines, as well as the horizon parameter. The real-time router reduces implementation complexity by sharing a single scheduler amongst the early and on-time traffic on each of the five output ports. Extensions to the scheduler architecture further reduce the implementation cost by trading space for time.

A. Integrating Early and On-Time Packets

To maximize link utilization and channel admissibility, each outgoing port should overlap packet scheduling operations with packet transmission. As a result, packet size determines the acceptable worst-case scheduling delay. Scheduling time-constrained traffic, based on delay or throughput parameters, typically requires a priority queue to rank the outgoing packets. Priority queue architectures introduce considerable hardware complexity [35–39], particularly when the link must handle a wide range of packet priorities or deadlines. For example, most high-speed solutions require $O(n)$ hardware complexity to rank n packets, using a systolic array or shift register consisting of n comparators [35, 40, 41]. Additional technical challenges arise in trying to integrate packet scheduling with bandwidth regulation [42], since the link cannot transmit a packet unless it has reached its logical arrival time.

To perform bandwidth regulation and deadline-based scheduling, the real-time router could include two priority queues for each of its five outgoing ports, as suggested by Table I. However, this approach would be extremely expensive and would require additional logic to transfer packets from the "early" queue to the "on-time" queue; this is particularly complicated when multiple packets reach their eligibility times simultaneously. In the worst case, an outgoing port could have to dequeue a packet from Queue 1 or Queue 3, enqueue several arriving packets to Queue 1

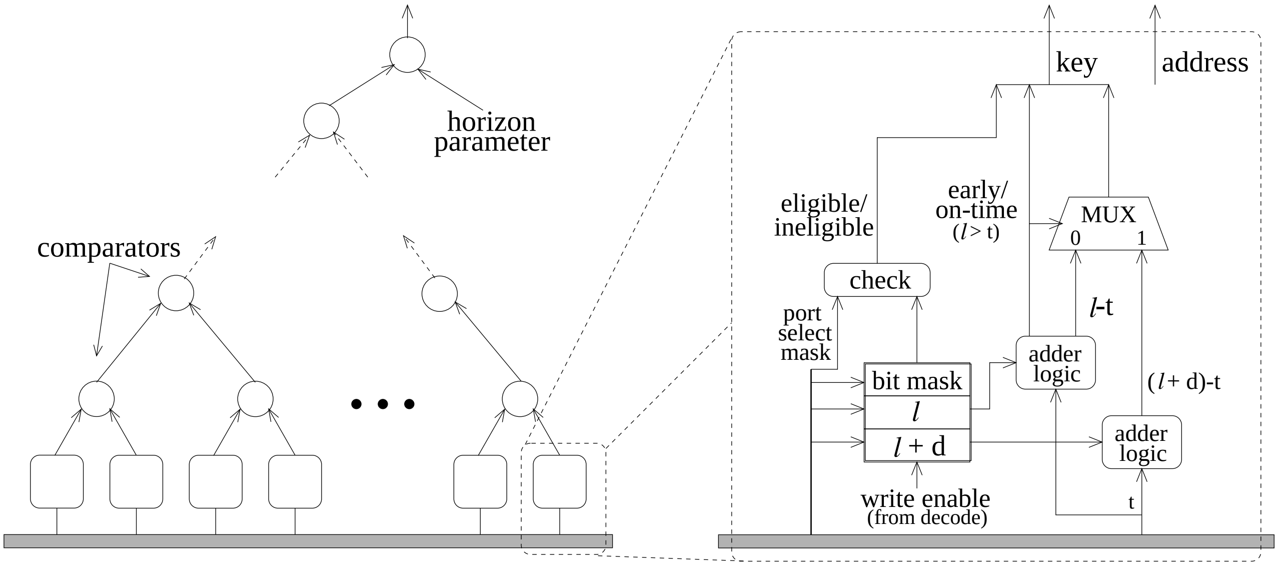


Fig. 6. **Comparator Tree Scheduler:** This figure shows the scheduling architecture in the real-time router. The leaf nodes at the base of the comparator tree stores a small amount of per-packet state information.

On-time:	0	0	$\ell(m) + d - t$
Early:	0	1	$\ell(m) - t$
Ineligible:	1	—	——

Fig. 7. **Scheduler Keys:** This figure illustrates how the real-time router assigns a key to each time-constrained packet awaiting transmission on an outgoing port. A single bit differentiates on-time and early packets; ineligible traffic refers to packets that are not destined to this port.

and/or Queue 3, and move a large number of packets from Queue 3 to Queue 1, all during a single packet transmission time. To avoid this complexity, the real-time router does not attempt to store the time-constrained packets in sorted order. Instead, the router selects the packet with the smallest key via a comparator tree, as shown in Figure 6. Like the systolic and shift register approaches, the tree architecture introduces $O(n)$ hardware complexity. For the moderate size of n in a single-chip router, the comparator tree can overlap the $O(\lg n)$ stages of delay with packet transmission.

To avoid this excessive complexity, the real-time router integrates early and on-time packets into a single data structure. Each link schedules time-constrained packets based on sorting keys, as shown in Figure 7, where smaller keys have higher priority. A single bit differentiates between early and on-time packets. 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. The packet keys are normalized, relative to current time t, to allow the scheduler to perform simple, unsigned comparison operations, even in the presence of clock rollover. Each scheduling operation operates independently to locate the packet with the minimum sorting key, permitting dynamic changes in the values of keys. The base of the tree computes a key for each packet, based on the packet state and the current time t, as shown in the right side of Figure 6; the base of the tree stores per-packet state information, whereas the packet memory stores the actual packet contents.

B. Sharing the Scheduler Across Output Ports

By using a comparator tree, instead of trying to store the packets in sorted order, the router can allow all five outgoing ports to *share* access to this scheduling logic, since the tree itself does not store the packet keys. As shown in Figure 6, 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 winner is an early packet that falls within the port's horizon parameter; if so, the port transmits this packet, unless best-effort flits await service.

Still, to share the comparator logic, the scheduler must operate quickly enough to overlap run-time scheduling with packet transmission on each of the outgoing ports. Consequently, the real-time router pipelines access to the comparator tree. With p stages of pipelining, the scheduler has a row of latches at $p-1$ levels in the tree, to store the sorting key and buffer location for the winning packet in the subtrees. Every few cycles, another link begins its scheduling operation at the base of the tree. Similarly, every few

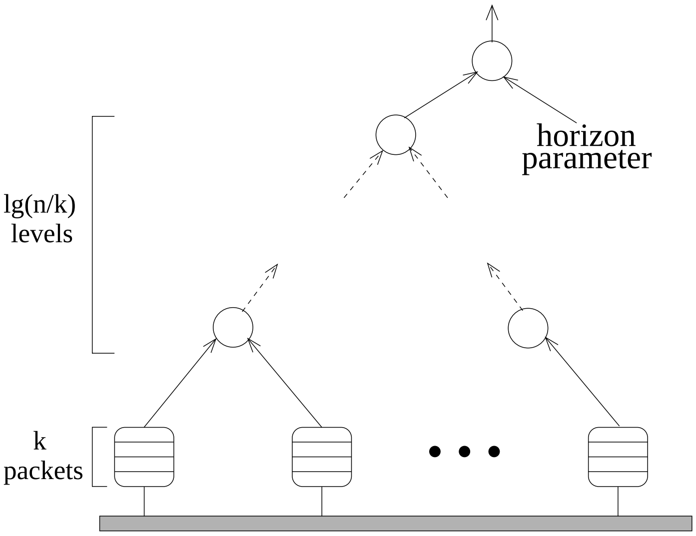


Fig. 8. **Logic Sharing:** This figure illustrates how the scheduler can trade space for time by sharing comparator logic amongst groups of k packets.

cycles, another link completes a scheduling operation and can initiate a packet transmission. As a result, the router staggers packet departures on the five outgoing ports. The necessary amount of pipelining depends on the latency of the comparator tree, relative to the packet transmission delay.

C. Balancing Hardware Complexity and Scheduler Latency

The pipelined comparator tree has relatively low hardware cost, compared to alternate approaches that implement separate priority queues for the early and on-time packets on each outgoing port. However, as shown in Section VI, the scheduler logic is still the main source of complexity in the real-time router architecture. To handle n packets, the scheduler in Figure 6 has a total of $2+\lg n$ stages of logic, including the operations at the base of the tree as well as the comparator for the horizon parameter. In terms of implementation cost, the tree requires n comparators and n leaf nodes, for a total of $2n$ elements of similar complexity. As n grows, the number of leaf nodes can have a significant influence on the bus loading at the base of the tree. Fortunately, for certain values of n, the comparator tree has low enough latency to avoid the need to fully pipeline the scheduling logic. This suggests that the scheduler could reduce the number of comparators by trading space for time.

Under this approach, the scheduler combines several leaf units into a single module with a small memory (e.g., a register file) to store the deadlines and logical arrival times for k packets, as shown in Figure 8. At the base of the tree, each of the n/k modules can *sequentially* compare its k sorting keys, using a single comparator, to select the packet with the minimum key; this incurs k stages of delay. Then, a smaller comparator tree finds the smallest key amongst n/k packets. As a result, the scheduler incurs

$$(k+1)+\lg\left(\frac{n}{k}\right)$$

stages of delay. Note that, for $k=1$, the architecture reduces to the comparator tree in Figure 6, with its $2+\lg n$ stages of logic. For larger values of k, the scheduler has larger arbitration delay but reduced implementation complexity. The architecture in Figure 8 has $2n/k$ comparators, as well as a lighter bus loading of n/k elements at the base of the tree. In addition, larger values of k allow the base of the tree to consist of n/k k-element register files, instead of n individual registers, with a reduction in chip complexity. With a careful selection of n and k, the real-time router can have an efficient, single-chip implementation that performs bandwidth regulation and deadline-based scheduling on multiple outgoing ports.

VI. Performance Evaluation

To demonstrate the feasibility of the real-time router, and study its scaling properties, a prototype chip has been designed using the Verilog hardware description language and the Epoch silicon compiler from Cascade Design Automation. This framework facilitates a detailed evaluation of the implementation and performance properties of the architecture. The Epoch tools compile the structural and behavioral Verilog models to generate a chip layout and an annotated Verilog model for timing simulations. These tools permit extensive testing and performance evaluation without the expense of chip fabrication.

A. Router Complexity

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, as shown in Table IV. The scheduling logic accounts for the majority of the chip area, with the packet memory consuming much of the remaining space, as shown in Table V. 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; the memory access latency is the bottleneck in this realization of the router. 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 match the memory and link throughputs, the comparator tree consists of a two-stage pipeline, where each stage requires approximately 50 nsec.

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 multiple leaves of the tree, as

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 IV

Router Specification: This table summarizes the architectural parameters and chip complexity of the prototype implementation of the real-time router.

Unit	Area	Transistors
Packet scheduler	34.02 mm^2	555025
Memory and control	5.97 mm^2	268161
Best-effort support	1.55 mm^2	45352
Connection table	0.65 mm^2	20966
Idle-address pool	0.35 mm^2	15600

TABLE V

Router Components: This table summarizes the area contribution and transistor count for the main components of the router.

discussed in Section V-C. Figure 9 highlights the cost-performance trade-offs of logic sharing, based on Epoch implementations and Verilog simulation experiments. As k increases, the scheduler complexity decreases in terms of area, transistor count, and power dissipation, with reasonable increases in scheduler latency. The results start with a grouping size of $k = 4$, since the Epoch library does not support static RAM components with fewer than four lines. (For $k = 1$, the graphs plot results from the router implementation in Table V, which uses flip-flops to store packet state at the base of the tree. The Epoch silicon compiler generates a better automated layout of these flip-flops than of the small SRAMs, resulting in better area statistics in Table V, despite the larger transistor count. A manual layout would significantly improve the area statistics for $k > 1$; still, the area graph shows the relative improvement for larger values of k.)

These plots can help guide the trade-off between hardware complexity and scheduler latency in the router implementation. For example, a group size of $k = 4$ reduces the number of transistors by 45% (from $555,025$ to $306,829$). The number of transistors does not decrease by a factor of four, since the smaller scheduler still has to store the state information for each packet; in addition, the scheduler requires additional logic and registers to serialize access to the shared comparators. Still, logic sharing significantly reduces implementation complexity. Larger values of k further reduce the number of comparators and improve the density of the memory at the base of the tree. Scheduler latency does not grow significantly for small values of k. For $k = 4$, delay in the comparator tree increases by just 67% (from 0.115 μsec to 0.192 μsec). The lower bus loading at the base of the tree helps counteract the increased latency from serializing access to the first layer of comparators and significantly reduces power dissipation.

B. Simulation Experiments

Since Verilog simulations of the full chip are extremely memory and CPU intensive, we focus on a modest set of timing experiments, aimed mainly at testing the correctness of the design. 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 10 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 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

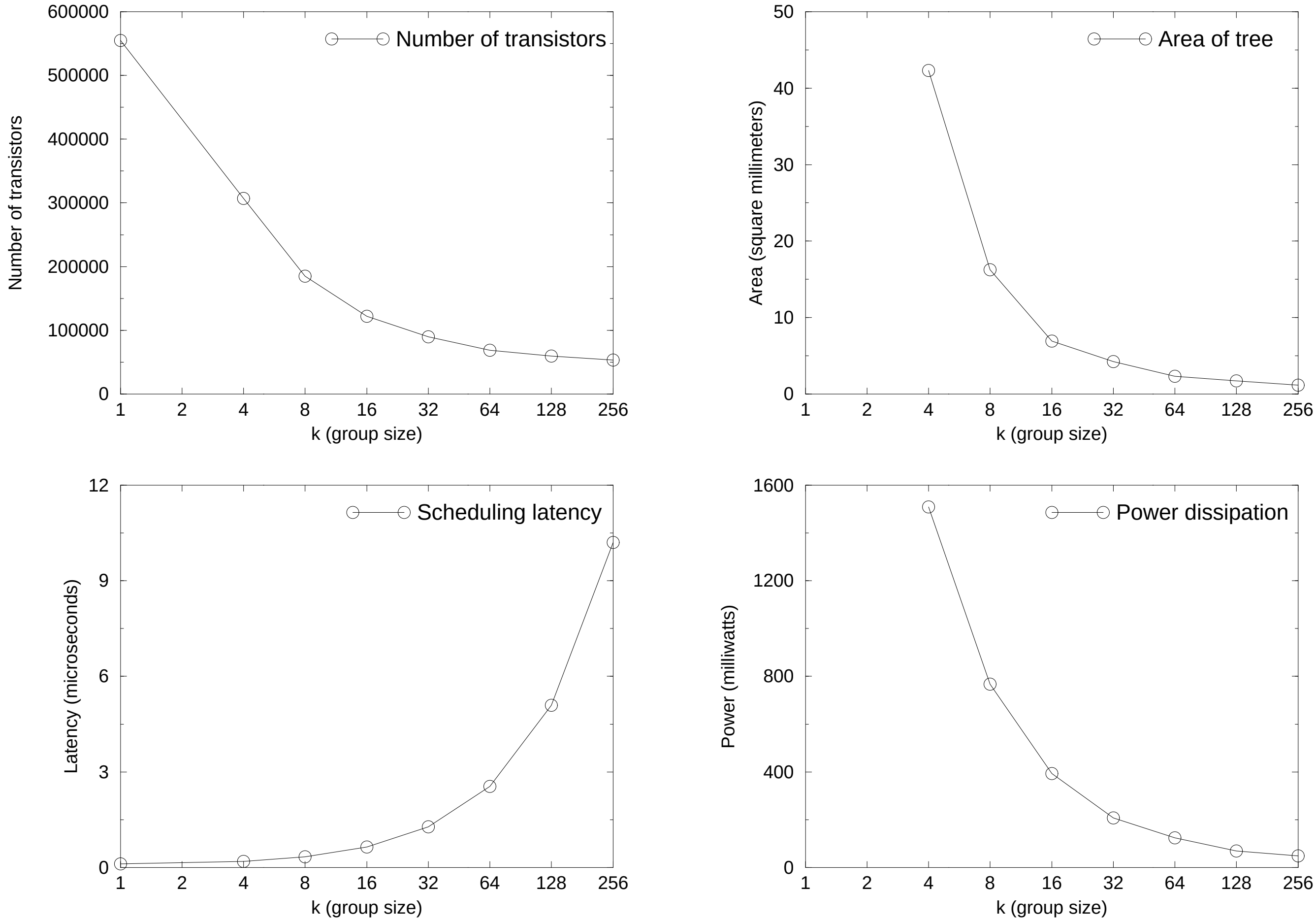


Fig. 9. **Evaluating Logic Sharing:** These plots compare different implementations of the comparator tree, with different group sizes k. As k grows, implementation complexity decreases but scheduler latency increases.

till its logical arrival time. Similarly, the link transmits each packet by its deadline, with best-effort flits consuming any remaining link bandwidth.

VII. Related Work

This paper complements recent work on support for real-time communication in parallel machines [2–7]. 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 [7]. 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 [23].

Although these mechanisms reduce variability in end-to-

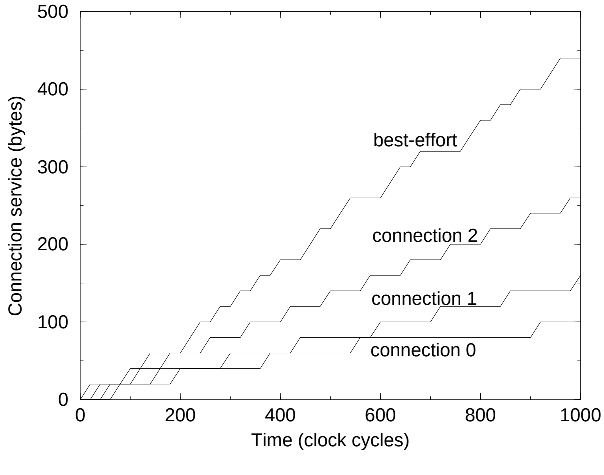


Fig. 10. **Timing Experiment:** This experiment evaluates a mixture of time-constrained and best-effort packets competing for access to a single outgoing link with horizon $h = 0$. The scheduler satisfies the deadlines of the time-constrained packets, while permitting best-effort flits to capitalize on any additional bandwidth.

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 [23]. 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 [4, 5].

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 [3]. 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 fine-grain packet priorities by increasing the number of virtual channels, at the expense of additional implementation complexity; these virtual channels incur the cost of additional flit buffers and larger virtual channel identifiers, as well as more complex switching and arbitration logic [32]. 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 [6] 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.

VIII. 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, switching, and flow control, 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 keys for logical arrival times and deadlines. Sharing scheduling logic and packet buffers amongst the five output ports permits a single-chip solution that handles up to 256 time-constrained packets simultaneously. Experiments with a detailed timing model of the router chip show that the design can operate at 50 MHz with appropriate pipelining of the scheduling logic. Further experiments show that the design can trade space for time to reduce the complexity of the packet scheduler.

As ongoing research, we are considering alternate link-scheduling algorithms that would improve the router's scalability. In this context, we are investigating efficient hardware architectures for integrating bandwidth regulation and packet scheduling [42]; these algorithms include approximate scheduling schemes that balance the trade-off between accuracy and complexity, allowing the router to efficiently handle a larger number of time-constrained packets. We are also exploring the use of the real-time router as a building block for constructing large, high-speed switches that support the quality-of-service requirements of real-time and multimedia applications. The router's delay and throughput guarantees for time-constrained traffic, combined with good best-effort performance and a single-chip implementation, can efficiently support a wide range of modern real-time applications, particularly in the context of tightly-coupled local area networks.

References

[1] D. Ferrari, "Client requirements for real-time communication services," *IEEE Communications Magazine*, pp. 65–72, November 1990.

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

[3] M. W. Mutka, "Using rate monotonic scheduling technology for real-time communications in a wormhole network," in *Proceedings of the Workshop on Parallel and Distributed Real-Time Systems*, April 1994.

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

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

[6] 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.

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

[8] S. Balakrishnan and F. Ozguner, "Providing message delivery guarantees in pipelined flit-buffered multiprocessor networks," in *Proceedings of the Real-Time Technology and Applications Symposium*, pp. 120–129, June 1996.

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

[10] 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.

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

[12] D. Verma, H. Zhang, and D. Ferrari, "Delay jitter control for real-time communication in a packet switching network," in *Proceedings of Tricom*, 1991.

[13] D. Ferrari and D. C. Verma, "A scheme for real-time channel establishment in wide-area networks," *IEEE Journal on Selected Areas in Communications*, vol. SAC-8, pp. 368–379, April 1990.

[14] H. Zhang and D. Ferrari, "Rate-controlled service disciplines," *Journal of High Speed Networks*, vol. 3, no. 4, pp. 389–412, 1994.

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

[16] L. Georgiadis, R. Guerin, V. Peris, and K. N. Sivarajan, "Efficient network QoS provisioning based on per node traffic shaping," *IEEE/ACM Transactions on Networking*, vol. 4, pp. 482–501, August 1996.

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

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

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

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

[21] 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.

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

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

[24] J. Rexford, J. Dolter, and K. G. Shin, "Hardware support for controlled interaction of guaranteed and best-effort communication," in *Proceedings of the Workshop on Parallel and Distributed Real-Time Systems*, pp. 188–193, April 1994.

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

[26] J. Rexford, W. Feng, J. Dolter, and K. G. Shin, "PP-MESS-SIM: A flexible and extensible simulator for evaluating multicomputer networks," *IEEE Transactions on Parallel and Distributed Systems*, pp. 25–40, January 1997.

[27] J. Duato and P. Lopez, "Bandwidth requirements for wormhole switches: A simple and efficient design," in *Proc. Euromicro Workshop on Parallel and Distributed Processing*, pp. 377–384, 1994.

[28] C. B. Stunkel *et al.*, "The SP2 high-performance switch," *IBM Systems Journal*, vol. 34, pp. 185–204, February 1995.

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

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

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

[32] 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.

[33] W. C. Lee, M. G. Hluchyj, and P. A. Humblet, "Routing subject to quality of service constraints in integrated communication networks," *IEEE Network Magazine*, pp. 46–55, July/August 1995.

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

[35] 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.

[36] D. Picker and R. D. Fellman, "VLSI priority packet queue with inheritance and overwrite," *IEEE Transactions on Very Large Scale Integration*, vol. 3, pp. 245–253, June 1995.

[37] J. Liebeherr, D. E. Wrege, and D. Ferrari, "Exact admission control for networks with bounded delay services," *IEEE/ACM Transactions on Networking*, vol. 4, pp. 885–901, December 1996.

[38] J. Rexford, A. Greenberg, and F. Bonomi, "Hardware-efficient fair queueing architectures for high-speed networks," in *Proceedings of IEEE INFOCOM*, pp. 638–646, March 1996.

[39] S.-W. Moon, K. Shin, and J. Rexford, "Scalable hardware priority queue architectures for high-speed packet switches," in *Proceedings of the Real-Time Technology and Applications Symposium*, pp. 203–212, June 1997.

[40] H. J. Chao and N. Uzun, "A VLSI sequencer chip for ATM traffic shaper and queue manager," *IEEE Journal of Solid-State Circuits*, vol. 27, pp. 1634–1643, November 1992.

[41] C. E. Leiserson, "Systolic priority queues," in *Proceedings of the Caltech Conference on VLSI*, pp. 200–214, January 1979.

[42] J. Rexford, F. Bonomi, A. Greenberg, and A. Wong, "Scalable architectures for integrated traffic shaping and link scheduling in high-speed ATM switches," *IEEE Journal on Selected Areas in Communications*, vol. 15, pp. 938–950, June 1997.

Jennifer Rexford received a B.S.E. degree in electrical engineering from Princeton University in 1991, and M.S. and Ph.D. degrees in computer science and engineering from the University of Michigan in Ann Arbor, in 1993 and 1996, respectively. Since 1996, she has been in the Networking and Distributed Systems center at AT&T Labs–Research in New Jersey. Her research interests include routing/signaling protocols, video streaming, and packet scheduling, with an emphasis on efficient support for quality-of-service guarantees. Her e-mail address is jrex@research.att.com.

John M. Hall received a B.S.E. degree in computer engineering from the University of Michigan, Ann Arbor, Michigan in 1996. He worked at Hewlett-Packard's Microprocessor Technology Lab, Fort Collins, Colorado working on the physical design of an IA-64 microprocessor and has recently returned to the University of Michigan graduate school majoring in electrical engineering. His e-mail address is hallj@umich.edu

Kang G. Shin is Professor and Director of the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor. He has authored/coauthored more than 460 technical papers (about 170 of these in archival journals) and numerous book chapters in the areas of distributed real-time computing and control, fault-tolerant computing, computer architecture, robotics and automation, and intelligent manufacturing. He has coauthored (jointly with C. M. Krishna) a textbook "Real-Time Systems," McGraw Hill, 1997. In 1987, he received the Outstanding IEEE Transactions on Automatic Control Paper Award for a paper on robot trajectory planning. In 1989, he also received the Research Excellence Award from The University of Michigan. In 1985, he founded the Real-Time Computing Laboratory, where he and his colleagues are investigating various issues related to real-time and fault-tolerant computing. He has also been applying the basic research results of real-time computing to multimedia systems, intelligent transportation systems, embedded systems, and manufacturing applications.

He received the B.S. degree in Electronics Engineering from Seoul National University, Seoul, Korea, in 1970, and both the M.S. and

Ph.D. degrees in Electrical Engineering from Cornell University, Ithaca, New York, in 1976 and 1978, respectively. From 1978 to 1982 he was on the faculty of Rensselaer Polytechnic Institute, Troy, New York. He has held visiting positions at the U.S. Air Force Flight Dynamics Laboratory, AT&T Bell Laboratories, Computer Science Division at the University of California at Berkeley, IBM T.J. Watson Research Center, and the Software Engineering Institute at Carnegie Mellon University. He also chaired the Computer Science and Engineering Division at The University of Michigan for three years beginning in January 1991. He is an IEEE Fellow, was the Program Chairman of the 1986 *IEEE Real-Time Systems Symposium* (RTSS), the General Chairman of the 1987 RTSS, the Guest Editor of the 1987 special issue of *IEEE Transactions on Computers* on real-time systems, a Program Co-Chair for the 1992 International Conference on Parallel Processing, and served on numerous technical program committees. He also chaired the IEEE Technical Committee on Real-Time Systems during 1991-1993, was a Distinguished Visitor of the Computer Society of the IEEE, an Editor of *IEEE Transactions on Parallel and Distributed Computing*, and an Area Editor of *International Journal of Time-Critical Computing Systems*.