Real-Time Communication in Point–to–Point Networks

Atri Indiresan, Kang G. Shin

Real-time Computing Laboratory
Dept. of Elec. Engr. and Comp. Sci.
The University of Michigan
Ann Arbor, Michigan 48109–2122.
{*atri,kgshin*}@*eecs.umich.edu*

Dilip D. Kandlur

IBM Research Division
T. J. Watson Research Center
Yorktown Heights, NY 10598
kandlur@watson.ibm.com

1 Introduction

Real-time systems are evolving from simple applications to more sophisticated ones with stringent performance and reliability requirements. Distributed systems are potentially well suited to meeting these requirements since they can provide features such as parallel computation, scalable performance growth, and graceful degradation in the presence of faults. Traditionally, such systems rely on either contention-based/time-division multiplexed access to a communication medium or completely-connected networks. While the latter is highly reliable, it is not scalable beyond a small number of processors. Contention-based networks have the further problem of potential for random delays in accessing the communication medium. A compromise between these two approaches are multicomputer systems with point–to–point interconnection networks. Such systems are scalable to large numbers of processors (100s or 1000s), and multiple disjoint paths between nodes make them robust to link and node failures. Examples of this kind of network include meshes and hypercubes.

Since real-time tasks have deadlines associated with them, we must provide guarantees to predict the performance of applications. Since the tasks are distributed over multiple nodes of the system, the total execution time includes both computation and communication times, and hence predictable communication latency is a necessary condition for providing any form of guarantee. In general, messages in point–to–point networks are subject to random delays due to contention for network bandwidth, buffer space and message processing bandwidth. If we can resolve this problem of unpredictability in communication, given the advantages of such systems in scalability and fault tolerance, they are potentially the most suitable solution to the requirements of future large scale real-time systems.

A theoretical model for real-time communication in a point–to–point network is described in [1] using a communication abstraction called a *real-time channel* as its basis. A real-time channel, or simply *channel*, is characterized by its source and destination, its performance requirements and traffic patterns, and resources including network bandwidth, buffer space and message processing bandwidth need to be reserved for it. This is because, in order to make any guarantees about the message delivery times, it is necessary to describe the characteristics of communication and reserve resources based on these characteristics.

The work reported in this paper was supported in part by the Office of Naval Research under grants N00014–92–J–1080 and N00014–J–91–1115.

2 Real-time Communication

Message traffic in a real-time system is of two types: real-time and non-real-time or best-effort. Latency of real-time traffic has to be predictable, because unpredictable delays in the delivery of messages can adversely affect the timely execution of tasks dependent on these messages. Predictable communication is difficult to achieve since network delays can be non-deterministic for numerous types of networks. In addition, though we make no guarantees about the latency of best-effort messages, it is desirable to avoid unduly penalizing their performance in the presence of real-time messages.

The real-time channel abstraction can be used to provide predictable performance for real-time message traffic. To establish a channel, it is necessary to specify the source and destination of the channel, the worst-case traffic patterns, and the maximum allowable end–to–end delay. The channel establishment procedure has to select a route through the network from source to destination, ensuring that adequate communication bandwidth, buffer space, and processing bandwidth are available at all intermediate nodes in the path. Link utilization for a channel is computed assuming the worst-case arrival time, and the message deadline minus its generation time as its period. A channel may be established if the necessary resources are available at each node of the route selected for the channel. The route and the worst-case delay for each link on the route are recorded in data structures which are used to set deadlines at each node along the channel for messages belonging to real-time channels. While details of the channel establishment procedure may be found in [1, 2], we briefly discuss the scheduling aspect.

There are several approaches to scheduling these messages, which can be categorized as fixed priority or dynamic priority algorithms. For example, Earliest Due Date (EDD) scheduling [3] is a dynamic priority algorithm, whereas rate monotonic scheduling [4] is a fixed priority algorithm.

Liu and Layland [4] have shown that EDD is optimal for preemptive scheduling of periodic tasks when the task deadline is equal to the beginning of its next period, and that a feasible schedule exists whenever the total utilization is less than one. However, there is no similar result (based on utilization alone) available when the task deadlines are not related to their periods. The main drawback of EDD scheduling is that computation of guarantees is difficult, since the priority of a task depends upon the relative order of arrival of the tasks.

Scheduling decisions can be based on a fixed (static) priority scheduling algorithm where messages are processed and transmitted in the order of priority. For example, in rate-monotonic scheduling, the priority assigned to a channel is related to the frequency of occurrence of messages on that channel. For any priority assignment scheme, if message arrivals on all the channels are assumed to be strictly periodic, we can determine whether the set of channels is schedulable. This is done by computing the worst-case response time for each message using a *critical time zone* analysis similar to the one used in [4], and verifying that the worst-case response time is less than the delay assigned to that channel. The details of this scheme for analysis can be found in [1].

3 Implementation

There are three main steps in the implementation of real-time channels: (i) the computation of a (feasible) route and required resources, (ii) the establishment of the route with the reservation of resources and bandwidth, and (iii) run-time support for routing and scheduling a mix of real-time and best-effort messages through the network. We now show how these can be achieved using the x-kernel [5] on the Network Processor (NP) of HARTS [6].

The x-kernel is designed for distributed system support, and provides facilities for implementing protocols like a uniform protocol interface, libraries to efficiently manipulate messages, and tools to configure and test different protocol stacks. We implement real-time channels as a set of protocol objects (using x-kernel terminology). The Real-Time Channel Protocol (RTCP) [1] is the protocol in the x-kernel protocol stack that implements the interface for the real-time channel communication scheme. The clients of this protocol are processes running on the Application Processors (APs).

Channel establishment has been dealt with elsewhere [1]. In brief, a separate protocol module called a *Network Manager* (NM) performs this service. Since requests to this server are serialized, data consistency for different channel establishment requests is assured. A request to the RTCP to establish a channel is passed on to the NM. If the request is successful, the channel may now send messages to the individual nodes along the route so that the necessary resources are reserved. In addition to resources such as buffers, at establishment time, for each channel, at the source and destination nodes, a process is created as a reserved resource to marshal messages through the protocol stack. It is expected that the context switching overhead will be less than the cost of creating and destroying a process for each message.

When we consider message scheduling, we encounter some difficulties with the x-kernel. The x-kernel uses a fixed-priority scheduling policy and there are some problems with priority based scheduling and the computation of guarantees when we consider multiple stage service, such as service for a message which has to traverse multiple links. The response time for the message varies depending upon the arrival time of other messages at a node. Therefore, even if messages are generated with a fixed inter-arrival time at the source, the inter-arrival time for the message at the next service point is not constant. Early arrivals are also possible due to burstiness in the message generation at the source. A message which arrives early at a node can cause a lower priority message to miss its deadline. One solution for this problem is to hold back the early arrival and not consider it for transmission until its scheduled arrival time. However, this scheme involves the setting and resetting of timers and is expensive to implement. Also, it implies that the message cannot make the best progress possible on lightly loaded links. If deadline scheduling is used, this problem can be handled more elegantly by proportionately increasing the deadline for an early arrival without missing the end–to–end deadline.

From the above discussion, we can see that there are three classes of messages: (a) real-time messages that arrive at a node in the expected time zone (queue 1), (b) best-effort messages (queue 2), and (c) real-time messages that arrive early (queue 3). Since messages in the third category need to be held back, we can transmit best-effort messages ahead of these, and behind "on-time" real-time messages. For the best-effort messages, a simple FIFO is adequate, but, for the real-time messages, we need to use EDD scheduling.

The most common, and most efficient way, to implement EDD scheduling is by using a priority queue, ordered by message deadlines. Using a heap, we may insert a message in $\Theta(\log n)$ time, and remove the next message to be transmitted in constant time [7]. When a message arrives, if it is a best-effort message, simply insert it into the second queue. If it is an early real-time message, insert it into the third queue, and if it is an "on-time" message, insert it into the first queue. Whenever the transmitter is free, transmit the message at the head of queue 1, if any, else messages from queue 2. When queue 1 is empty, check the messages in queue 3 to see if any of them have become current, and, if so, they may be inserted into queue 1.

Conceptually, we do need such a scheme, but, in practice, there may be some problems. Though $\Theta(\log n)$ time may be a fairly small number, the cost of each step compared to the cost of message transmission may be large since messages are typically very short, and message transmission time may be less than the scheduling time, making the scheduling rather than the communication bandwidth the bottleneck. Keeping such an eventuality in mind, we propose a hardware priority queue that will insert a message in a priority queue in a small constant number of clock cycles [8]. This design has a large shift register containing message deadlines, and when a new message arrives, its deadline is compared to those of the messages in the shift register, allowing it to be inserted in the appropriate place. Removing a message after transmission simply involves a shift operation.

4 Summary and Conclusions

We have discussed the concept of a real-time channel, and explained how it could be used to provide guaranteed message latencies in multi-hop interconnection networks. We have explored the issues in setting up such channels and providing run-time scheduling for them. Software scheduling for real-time messages may be too slow to handle high message traffic, and so we have proposed a hardware scheduling mechanism.

Several protocols have been implemented on the NP of HARTS, including the hexagonal mesh link-level protocol and a request-reply protocol for RPC. A Network Manager accepts requests for the establishment of channels and computes the routes and the resource requirements for the channels.

We are in the process of implementing the software run-time scheduling algorithm, and plan to characterize its performance to determine the workload conditions under which the scheduling overhead becomes unacceptable. The basic architecture of the hardware scheduler has been completed, and we are considering integrating it in HARTS to overcome the software scheduling bottleneck.

We plan to evaluate different scheduling algorithms, with different tradeoffs in performance and real-time characteristics. For instance, if message generation frequency is low, EDD and FIFO scheduling will be almost identical, and so we need not pay the EDD overhead. Other aspects of interest are the impact of real-time messages on the performance of best-effort messages. Real-time channels are very conservative in the sense that they assume worst-case traffic conditions. It would be interesting to explore whether a more optimistic resource allocation may be used without compromising the real-time requirements. This would involve making predictions of expected usage, and allocate fewer resources for each channel. Since guarantees are no longer strict, it would be necessary to monitor the system, and notify users when latency bounds are (consistently) violated.

Further, the current version of the real-time channel [1] provides guarantees in the absence of failures. In the event of failures, rerouting messages around the faulty links and nodes may cause the messages to miss their deadline. An alternative is the concept of group channels, where we send replicated messages. This not only increases the cost of communication, but raises issues of group membership, message ordering and failure semantics. We plan to explore this problem, and evaluate possible tradeoffs in cost and reliability of the various potential alternatives.

References

[1] D. D. Kandlur and K. G. Shin, "Design of a communication subsystem for HARTS," Technical Report CSE–TR–109–91, CSE Division, Department of EECS, The University of Michigan, October 1991.

[2] D. D. Kandlur, K. G. Shin, and D. Ferrari, "Real-time communication in multi-hop networks," in *Proc. Int. Conf. on Distributed Computer Systems*, pp. 300–307, May 1991.

[3] M. L. Dertouzos, "Control robotics: The procedural control of physical processes," in *Proceedings IFIP Congress*, pp. 807–813, 1974.

[4] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, January 1973.

[5] N. C. Hutchinson and L. L. Peterson, "The x-Kernel: An architecture for implementing network protocols," *IEEE Trans. Software Engineering*, vol. 17, no. 1, pp. 1–13, January 1991.

[6] K. G. Shin, "HARTS: A distributed real–time architecture," *IEEE Computer*, vol. 24, no. 5, pp. 25–36, May 1991.

[7] D. W. Jones, "An empirical study of priority-queue and event-set implementations," *Communications of the ACM*, vol. 29, no. 4, pp. 300–311, April 1986.

[8] A. Indiresan and Q. Zheng, "Design and evaluation of a fast deadline scheduling switch for multicomputers," RTCL working document, December 1991.