A Dynamic Timeout Scheme for Wormhole Routing Networks

Po-Chi Hu
Lucent Technologies Inc.
200 Schulz Drive
Red Bank, NJ 07701

Leonard Kleinrock[†]
Department of Computer Science
University of California, Los Angeles
Los Angeles, CA 90095-1596

Abstract

In a previous paper [1], we proposed and analyzed a timeout scheme to alleviate network congestion and thus improve the throughput for a wormhole routing network in local area network (LAN) environments. This timeout scheme was proved to be effective, but the optimal timeout value varies with packet size, propagation delay, and other network parameters. To tune the timeout value automatically, two dynamic timeout methods are presented in this paper. The first method, called *"Immediate Timeout Or Wait"* (ITOW), compares the costs for timeout and waiting. Thus, it decides to reject a worm immediately or to allow the worm to wait until the timeout occurs. The second method, called *"Cost Equilibrium Point"* (CEP), sets the timeout value to the point where the timeout and waiting costs are the same, thereby determining the timeout value automatically. From simulation, the results show that the first method simplifies the choice of the timeout value and the second method performs well if a cost factor for timeout is given properly. Both methods improve the network throughput significantly.

1 Introduction

Wormhole routing is a simple, low-cost switching scheme often used for supercomputer interconnections. It has the merits of low latency, low cost, and simple implementation. Recently wormhole routing has been used as the switching scheme for Local area networks (LANs). One such effort is Myricom's *Myrinet* [2], which has been adopted as the LAN infrastructure for the Supercomputer SuperNet (SSN), a research project being conducted at UCLA, JPL and Aerospace Corp. [3, 4].

1.1 Wormhole Routing

Wormhole routing was developed from the earlier idea of *cut-through switching* [5], and was first introduced in [6]. A wormhole routing network is composed of several switches and hosts. Usually, wormhole routing switches have relatively small buffers. As opposed to store-and-forward switching, as soon as a packet header (or its routing information) is received, this packet is forwarded to the next switch before it is completely received; however, if the outgoing link to the next switch is busy serving another packet, then the

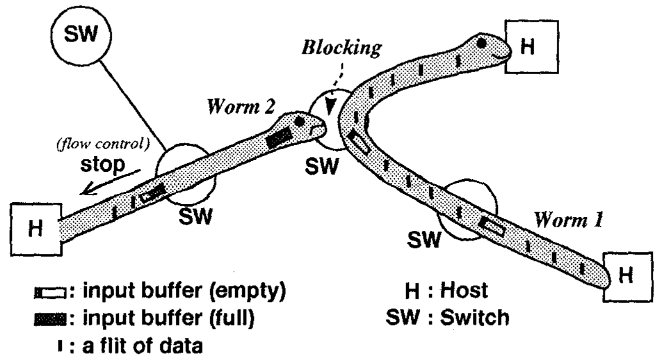

Figure 1: *An illustration of wormhole routing.*

packet is blocked and resides in the network (see figure 1) until the outgoing link is available. In this case, called *blocking*, the switch must inform the previous up-stream switch to stop transmission (i.e., it exercises *back-pressure flow control*) due to the limited size of buffers, as shown in figure 1. A packet (which is also called a *worm*) may be buffered along a chain of switching nodes when blocked. With wormhole routing, deadlocks are possible unless a deadlock-free routing strategy is employed. We measure packet length by *flits*, which is the amount of data that can be transmitted in one time unit. For example, the 640Mbps Myrinet [2] has one byte per flit lasting 12.5ns. A survey of wormhole routing can be found in [7].

1.2 Timeout Reset

Wormhole routing exhibits very low network latency through its use of cut-through switching [5]. However, a high-speed LAN requires not only low latency but also very high throughput, which is not easy to achieve with wormhole routing because of the blocking problem. Blocking occurs when there are two packets contending for the same output link; one of them has to be stalled, which consequently reduces the efficiency of links that are occupied by the blocked packet. This degrades the achievable network throughput. To overcome this throughput limitation, a deterministic timeout scheme was proposed in our earlier paper [1]. Whenever a worm head reaches a switch, a timer starts counting how long this worm resides at this switch while waiting for its outgoing link to become available (at

[†]This work was supported by the Advanced Research Projects Agency, ARPA/CSTO, under Contract DABT63-93-C-0055 "The Distributed Supercomputer Supernet — A Multi Service Optical Intelligent Network".

which time it advances to the next switch or host node). If this "residence time" exceeds a timeout threshold, then a timeout event is triggered; a switch at which timeout occurs will then clear all buffers occupied by this worm and will issue a timeout reset signal backward to the upstream node from which this worm came. A switch which receives a timeout reset signal will pass this signal further upstream and will also free the outgoing link and any buffers occupied by this timed-out worm. This process continues until the timeout reset signal reaches the source host where the worm was generated. (We assume that a switch can always send the timeout reset signal upstream even if the tail of the worm has already left this switch). The source host, after receiving the timeout reset signal, will stop the transmission of this worm if the transmission is still in progress, and will insert the worm back into the tail of this host's packet queue so that it will be retransmitted later.

In the deterministic timeout scheme, all timeout values are the same, no matter how far the worm has traveled or how large the worm is. However, the timeout value may be set dynamically to increase the network throughput and to tune the timeout value automatically. This is called ***dynamic timeout***, and is presented in this paper.

To tune the timeout value automatically, switches must be able to collect measurements of some basic parameters and to predict their future values. To maintain a simple structure of switches, only primitive functions, such as recording the link occupancy time for a worm and counting the number of worms currently waiting, are considered. More complex and intelligent functions, such as counting the number of worms blocked along the path and passing this information to other switches, are not examined in this paper.

1.3 The Simulation Experiment

In this paper, we test the performance of two dynamic timeout schemes by simulation. To make the simulation study less complicated, we use a *torus* as the network topology due to its nice symmetry property. We assume that each switch (SW) has eight in/out ports and four of them are connected to hosts (H). A 3×3 example is shown in figure 2, in which there are 9 switches and 36 hosts.

The simulator performs *discrete-event* simulations at the flit level with the following assumptions:

- Worms are generated as a Poisson process, and their sizes have an exponential distribution.
- Worm generation rates are identical at all hosts. Moreover, the distance to a host is uniformly chosen from among all feasible distances. Using the 3×3 torus network as an example, we make one-third of the traffic two-hops long, another one-third three-hops long, and the last one-third four-hops long, (similarly for 5×5 and 7×7 networks). Hosts at the same distance from the source are selected as the destination according to a uniform distribution.
- All possible shortest paths are equally chosen by the routing procedure.

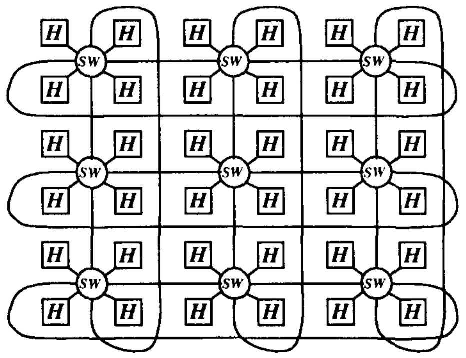

Figure 2: *An example of network configuration:* 3×3 *torus topology.*

- The bandwidth consumed by flow control and timeout signals is negligible.
- *First-come-first-serve* (FCFS) queueing discipline is used for resolving link contention.

In all simulations, the link propagation delay is set to 10 units of time, which corresponds to a link length of 22.5 meters in Myrinet[§].

In the following section, we first present the assumed primitive functions, and explain how they can be implemented. In section 3, we discuss the methods for estimating the costs for timeout and waiting. With these estimated costs, two dynamic timeout schemes are proposed and investigated in section 4. The conclusion is in section 5.

2 Assumed Switch Functions

To make the dynamic timeout practical for a low-cost wormhole routing switch, only some simple and local functions are assumed. These functions are:

- Estimate the average *link occupancy time*, which is the time interval that a served worm holds this link. It is also referred as the *link holding time*.
- Count how many worms are waiting for a particular link. This information is required to estimate the *link waiting time*, which is the time interval from when a worm arrives until it attains this output link. The link waiting time is actually the worm's blocking time at this switch.
- Retrieve and update the information carried at the worm head, such as worm size and the distance that the worm has traveled.

2.1 Estimating the Average Link Occupancy Time

To estimate the average link occupancy time, a switch first needs to be able to record the link occupancy time of a

[§]The *myrinet* link bandwidth is 640Mbps. Each flit is one byte of data, which gives us the time unit, 12.5ns. However, the propagation velocity in a Myrinet cable is about 0.6c, where c is the speed of light. Hence, $0.6c \times 12.5ns = 22.5m$.

worm. When a worm seizes an output link, a timer associated with this output link is reset, and keeps counting until the tail of the worm leaves this output link or it receives a timeout signal. Let x_i be the recorded link occupancy time of the ith worm on a particular output link. Then, the average link occupancy time for this link is estimated as,

$$O_i = \frac{(k-1)O_{i-1} + x_i}{k} \tag{1}$$

where O_i denotes the estimated average link occupancy time after the ith worm is served. k is a parameter we can select; it represents the weight of the newly recorded link occupancy time.

Equation (1) has the property that the estimated link occupancy time adjusts as the network condition changes, and the change rate is determined by the parameter k. Also, it has the advantage of only requiring two variables, O_i and x_i. Without retaining all link occupancy times of past worms, it is easy to implement.

2.2 Counting the Number of Waiting Worms

This can be easily accomplished by checking the headers of worms at the heads of all input buffers. Counting is initiated when a worm comes to the head of an input buffer and wishes to know its estimated waiting time. Alternatively, a counter associated with each output link may be used to store the number of waiting worms. This counter is updated when a worm arrives to the head of an input port and points to this output link (add one to the counter), or when the served worm releases this output link (decrease one from the counter).

2.3 Collecting the Worm's Information

Information such as worm size, the distance that it has traveled, or the distance that remains to its destination have effects on the optimal timeout value. This information can be derived quickly if they are in the worm's header, and if so, this facilitates updating the worms at each hop. Certainly, reading all of this information adds some complexity, but not much. It only requires a switch to read a few more bytes of the worm head before it determines the timeout value. In addition, updating the worm header simply involves incrementing or decrementing the distance counter that keeps track of how far the worm has traveled or how much further it must go.

3 Cost Estimation

With the above described information collected, costs for timeout and waiting can be estimated. Cost is defined as the number of flits that could have been transmitted on links but were not due to timeout or waiting. Based on these estimated costs, a switch can determine the reaction to an arriving worm.

First we have to determine the cost for a timeout. Similarly, we must determine the cost for waiting. When a timeout occurs, a blocked worm is rejected and the links it holds are freed. Consequently, timeout both saves bandwidth, and also wastes bandwidth that has been utilized by the timeout-rejected worm. This waste of bandwidth is the cost for timeout. Thus, C_T, the cost for timeout is estimated as (see figure 3):

$$C_T = \alpha T_h D \tag{2}$$

where T_h denotes the average duration of time that a link is held by the worm, D is the distance (number of hops) that the worm head has traveled, and α is a constant used to include other cost factors, such as the blocking time at each hop and the timeout-retransmissions that occur prior to this hop.

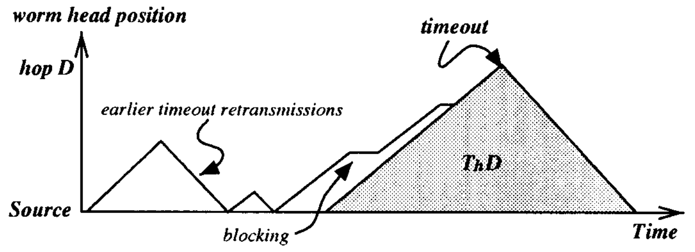

Figure 3: *An illustration of timeout cost.*

To be more specific, the cost for timeout is illustrated in figure 4, and calculated using the following equations:

$$N_l = \frac{L}{\tau_p} \tag{3}$$

$$C_T = \begin{cases} \alpha\left[LD - \frac{1}{4}N_l L\right] & \text{if } D > \frac{N_l}{2} \\ \alpha\tau_p D^2 & \text{otherwise} \end{cases} \tag{4}$$

where τ_p is the propagation delay on each link, L (time units) is the worm size divided by link capacity, and N_l is the number of links that the worm can spread over.

Similarly, the cost for waiting, C_W, is the amount of bandwidth wasted while the worm is waiting. It is derived as (see figure 5):

$$C_W = \begin{cases} T_l N_w N_l & \text{if } D > N_l \\ T_l N_w D & \text{otherwise} \end{cases} \tag{5}$$

where T_l is the average link occupancy time estimated in section 2.1 (i.e., O_i), and N_w is the number of worms waiting for the same output link.

4 Timeout Strategies

We tested two strategies that determine the timeout value. The first one is called "*ITOW*" (*Immediate Timeout Or Wait*), and the second one is called "*CEP*" (*Cost Equilibrium Point*).

4.1 Immediate Timeout Or Wait (ITOW)

With this strategy, timeout occurs immediately if a switch finds that the arrival worm has the estimated cost for timeout less than for waiting. The motivation of this immediate rejection comes from the following scenario. Consider

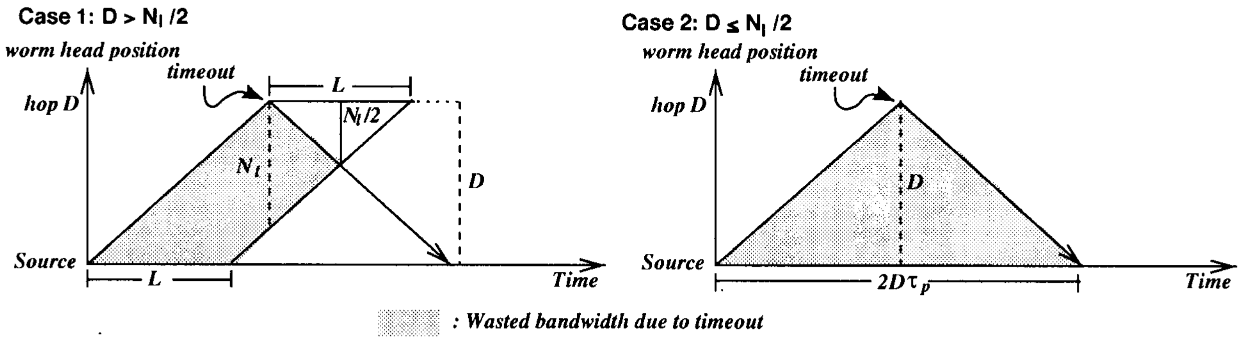

Figure 4: *A detailed illustration of timeout cost.*

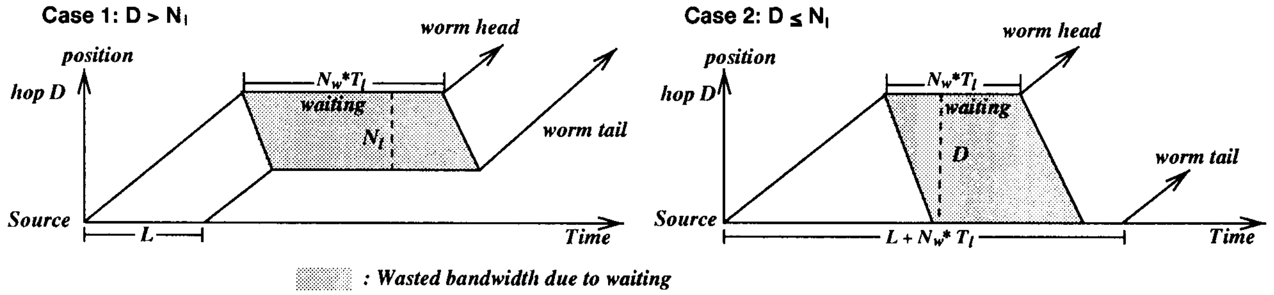

Figure 5: *An illustration of waiting cost.*

the situation that a blocked worm will get timed-out eventually. Instead of rejecting the worm after the timeout period, why not reject the worm immediately? To reject a worm sooner saves the bandwidth wasted by the waiting worm. If the estimated cost for waiting is less than for a timeout, the blocked worm is allowed to wait up to the timeout period. In this case, a timeout is still required to prevent inaccurate cost prediction and deadlock.

By simulation, the performance of this strategy is presented in figures 6 and 7. Figure 6 shows that the network throughput is quite stable even though the timeout value varies from 10 to 100. In addition, this scheme outperforms the deterministic timeout. Clearly, ITOW improves the network throughput significantly and eases the choice of timeout value.

Figure 7 shows the network throughput with different timeout cost factors. It appears that the optimal cost factor is about 6.0 for the 7×7 and 9.0 for the 5×5 torus case.

4.2 Cost Equilibrium Point (CEP)

CEP lets the timeout value be determined automatically by the switches. CEP sets the timeout value, τ_t, at the point where the estimated cost for timeout is equal to the estimated cost for waiting (hence the name, *Cost Equilibrium Point*). Thus,

$$\tau_t = \begin{cases} \frac{C_T}{N_l} & \text{if } D > N_l \\ \frac{C_T}{D} & \text{otherwise} \end{cases} \qquad (6)$$

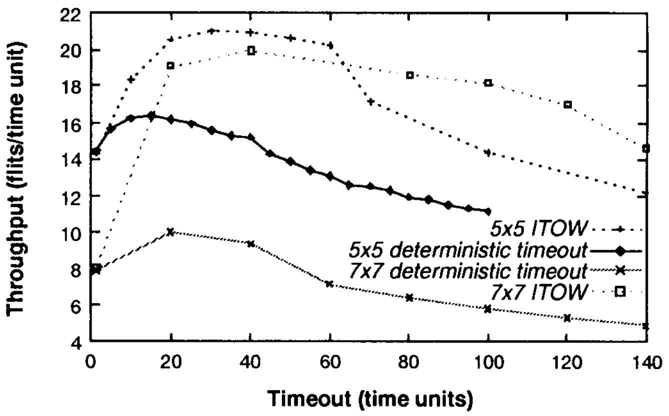

Figure 6: *Throughput vs. timeout (torus, average worm size = 50 flits, propagation delay = 10 time units, α = 10).*

This timeout setting tries to limit the worst case to have its cost be less than twice the optimal, no matter what the waiting time distribution is for the output link. However, the cost factor, α still needs to be chosen properly to optimize network performance.

By simulation, figure 8 shows that network throughput is optimized when the timeout cost factor is 3.0 for the 7×7 torus case. A similar result is found for the 5×5 case, as

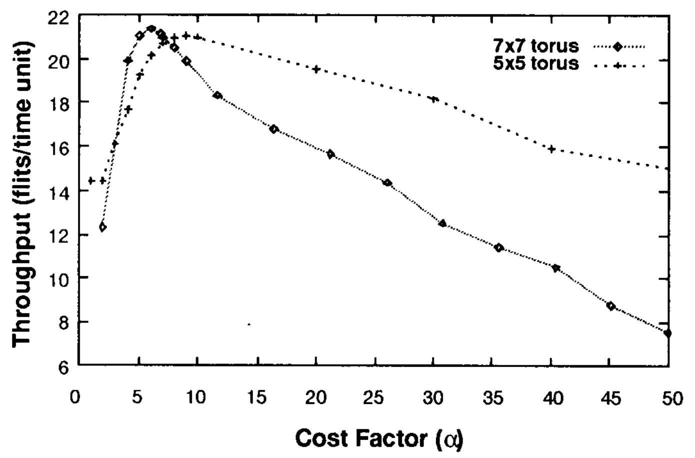

Figure 7: *Throughput vs. cost factor α (7×7 torus, average worm size = 50 flits, propagation delay = 10 time units, timeout = 40 time units).*

shown in figure 8.

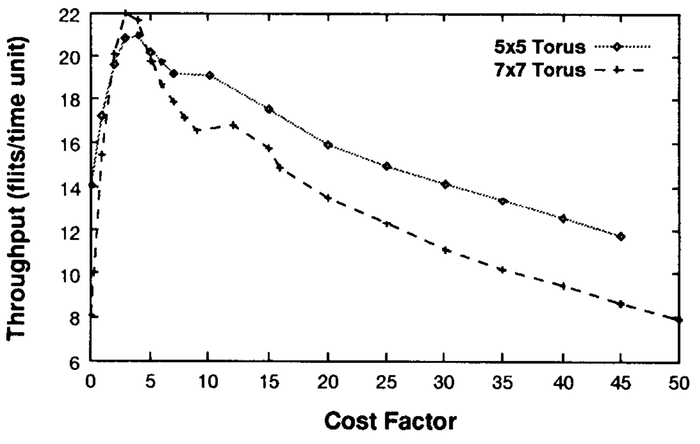

Figure 8: *Throughput vs. cost factor α (torus, average worm size = 50 flits, propagation delay = 10 time units).*

Comparing CEP and ITOW, we find that CEP outperforms ITOW slightly if the cost factor, α, is chosen properly. However, CEP is more sensitive to the cost factor than ITOW. Both improve the network throughput over the deterministic timeout significantly.

5 Summary and Future Work

In this paper, we have presented methods to perform timeouts dynamically. Two strategies are discussed: ITOW (Immediate Timeout Or Wait) and CEP (Cost Equilibrium Point). The results show that ITOW eases the choice of the timeout value and is less sensitive to the cost factor than CEP. However, CEP sets the timeout value automatically without the knowledge of the average link occupancy time. Both improve the network throughput significantly when compared to the deterministic timeout.

Further investigation is necessary. First, the cost factor, α should be refined for different network environments. As the worm travels more hops, the number of retransmissions required for the worm to reach the timeout node will increase. This implies that the cost factor should increase as the worm travels farther. One must also study the effect of k, the weighting for recently recorded link occupancy times. If the link occupancy time is highly correlated between subsequent worms, k should be decreased to reflect the changing of the average link occupancy time. On the other hand, if link occupancy time is mutually independent, k should be larger so it will not be affected by a single case. A good choice of k could make the cost prediction more accurate and therefore, improve the network performance.

References

[1] Po-Chi Hu and L. Kleinrock. "A Queueing Model for Wormhole Routing with Timeout". In *Proceedings of the 4th International Conference on Computer Communications and Networks*, pages 584-593, Las Vegas, NV, U.S., September 1995.

[2] C. Seitz, D. Cohen, and R. Felderman. "Myrinet—A Gigabit-per-second Local-Area Network". *IEEE Micro*, 15(1):29-36, February 1995.

[3] et. al L. Kleinrock. "The Supercomputer Supernet: A Scalable Distributed Terabit Network". *Journal of High Speed Networks: special issue on Optical Networks*, 4(4):407-24, 1995.

[4] et. al L. Kleinrock. "The Supercomputer Supernet Testbed: A WDM Based Supercomputer Interconnect". to appear in *IEEE JSAC/JLWT* joint special issue on Multiple Wavelength Optical Technologies and Networks, 1995.

[5] P. Kermani and L. Kleinrock. "Virtual cut-through: A New Computer Communication Switching Technique". *Computer Networks*, 3:267-289, 1979.

[6] C. Seitz *et al.* "The Hypercube Communications Chip". Technical report, Dep. Computer Science, California Inst., March 1985. Display File 5128:DF:85.

[7] L. M. Ni and P. K. McKinley. "A Survey of Wormhole Routing Techniques in Direct Networks". *Computer*, pages 62-76, February 1993.