Distributed Connection Management for Real-Time Communication over Wormhole-Routed Networks

Sharad Sundaresan Riccardo Bettati
Department of Computer Science
H.R. Bright Building
Texas A&M University
College Station, TX 77843-3112
tel: (409) 845-5469
fax: (409) 847-8578
{sharad,bettati}@cs.tamu.edu

Technical Report 96-021

December 12, 1996

Abstract

Wormhole networks provide a very-high-speed communication medium that is well suited for a large number of bandwidth demanding applications. Unfortunately, the lack of buffering in the switches causes blocked packets to transiently monopolize arbitrary large portions of the network, thus making it difficult to give real-time guarantees when connections contend for communication links. We present an efficient, fully distributed admission control scheme for time-critical connections over wormhole networks. Such a distributed scheme offers advantages in terms of scalability and easy integration into hybrid networks, over a corresponding centralized scheme. We compare this scheme to a centralized algorithm in a suite of simulation experiments and show that its performance is encouraging.

1 Introduction

The increased development and use of networked real-time applications in distributed multimedia, remote laboratoring, and distributed virtual reality has generated a large amount of interest in the development of real-time communication protocols to support such applications. Such protocols provide real-time guarantees for message deliveries to the network clients, typically in terms of bounded delay and jitter. This in turn enables the applications to meet their end-to-end timing requirements. In order to provide such guarantees on message delivery, resources in the network must be appropriately allocated. The admission control component of the protocol must ensure that enough resources are available in the system so that a new connection can in fact be guaranteed the required performance. Such admission control protocols have been extensively studied for packet-switched networks [4, 9, 12, 11].

Due to the increasing bandwidth requirements in a large number of applications (such as incorporation of uncompressed video streams in control loops for remote laboratoring), a number of very-high speed networking technologies are currently under investigation for their applicability to support hard-real-time communication, such as ATM [2], FDDI [1], HIPPI [3]. We are investigating the applicability of wormhole-routed network technologies for this class of applications.

As exemplified by recently developed technology, wormhole-routed networks are a promising approach for high-bandwidth, low-latency communication for small and medium-sized networks. For example, Myrinet switches [6, 7] have a link bandwidth of 1.28Gb/s with a worst-case routing latency of 550ns for a 8×8 switch. In the following discussion, we will be using Myrinet as a reference for the system model.

A well known problem with wormhole-routed packet networks is the potentially large amount of blocking that packets can experience because of contention for links. Because of the very limited amount of buffering in such networks, blocked packets "remain" in the network and keep using network resources. Thus, they may in turn cause other packets to be blocked. This may affect a large number of packets over a large portion of the network. Proper connection management strategies and appropriate protocols must be devised that ensure that blocking of packets due to link contention is bounded. Zhao *et. al* [5] have developed a transmission-control scheme that regulates the rate of connections at the entrance to the network. A worst-case achievable utilization of 50% could be proved, allowing for a simple admission control scheme: new connections are accepted as long as the utilization of the network does not exceed the worst-case achievable utilization.

In this paper we propose a distributed admission control scheme with the following two objectives:

1. Exploit the concurrency available in the network. Not always do connections conflict for links. With appropriate routing, network utilizations that go far beyond the bounds described in [5] can be achieved.

2. The admission control scheme should be distributed and affect only the switches that are on the route of the connection. Node failures and network partitions are to affect the admission process only if they happen along the route of the connection.

The paper is organized as follows. In Section 2 we describe our model of packet transmission over wormhole-routed networks and derive simple worst-case bounds for the time which packets can be blocked during transmission. We also describe the basic traffic model used in the protocol and how the admission of a new connection affects existing connections. From this we derive a general admission control scheme. In Section 3, we describe our distributed admission control scheme based on residual time. In Section 4, we describe the protocol for the implementation of the distributed algorithm. Section 5 describes simulation results which illustrate that our scheme compares favorably against centralized schemes. We will conclude and give an outlook on future research in Section 6.

2 Real Time Communication over Wormhole Routed Networks

In the following we assume a Myrinet network of general topology, consisting of *hosts* and *switches* and pairs of unidirectional *links*. The links forward arbitrary-length packets, which are routed through the switches in wormhole manner. As reflected by the name, the process of routing and transmission of a packet is like the movement of a worm. Whenever the header of a packet reaches a switch, the remaining portion of the packet is forwarded to the desired outgoing port (if available) as soon as the header is decoded, without waiting for the entire packet to reach the switch. Longer packets can therefore extend over several switches. If the desired outgoing port is not available because it is being used for the transmission of another packet, the header of the packet blocks.

Myrinet switches are *perfect*, meaning that packets do not conflict at a switch unless they are directed to the same outgoing link. There may be as many packets simultaneously traversing the switch as there are outgoing links. If more than one packet must wait, they are served in round-robin manner.

A simple hop-by-hop STOP/GO flow control mechanism (which acts as *backpressure flow control*) ensures that the storing of a blocked packet blocks at the upstream switches as well. Since only a small buffer is associated with input ports at each switch, the packet can easily be blocked up to the sender.

In order to streamline the packet processing at switches, *source routing* is used. A source host determines the route to a destination (a list of switches and output ports) and places it into the header of the packet. When the header arrives at the switch, the router reads the information and determines the desired outgoing port. Then it is forwarded through the specified output port to the next switch.

In conjunction with its flexible point-to-point topology, this results in a low-cost but scalable networking technology that provides guaranteed delivery of packets at very high speed and low latency.

Figure 1 shows a snapshot of a wormhole network with a number of packets in traffic. In this example, the network consists of six hosts $H_1, \ldots, H_6$, which are connected by nine 4×4 switches $S_1, \ldots, S_9$, forming a mesh. In this example, Packet P_2 is being blocked on the link between S_5 and S_8 by Packet P_1. Packets P_3 and P_2, on the other side, do not affect each other, although they use the same Switch S_4. This example illustrates also how a packet can be *indirectly blocked* by another packet: although P_1 and P_5 do not directly contend for a link, P_1 indirectly blocks P_5 by blocking P_2, which in turn blocks P_5 on the link from S_4 to S_5.

Once the head of the packet reaches the destination, the remainder can be sent at potentially full link bandwidth (of 640 Mb/s or 1.28Gb/s in the case of Myrinet). In order to bound packet delivery delay due to blocking during path setup, however, proper connection acceptance and connection management strategies must be devised. In the following, we assume that connections adhere to a periodic traffic specification. Each connection is of type $C_i = (R_i, e_i, p_i)$, where R_i specifies the route followed by packets of C_i, e_i defines the worst-case message length (in terms of transmission time), and p_i specifies the worst-case interarrival time of packets of C_i.

We assume without loss of generality that the packets of Connection C_i enter the network at Switch S_K and are routed along the sequence of switches $S_K, \ldots, S_{k+1}, S_k, S_{k-1}, \ldots, S_1$. For convenience we call the sender host S_{K+1} and the receiver host S_0. Assuming a round-robin service

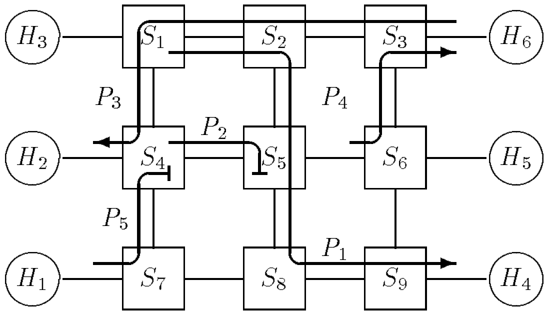


Figure 1: Snapshot of a Network with Packets in Transit

policy in the switches, we can determine the *worst-case blocking time* $B_{i,k}$ experienced by a packet of connection C_i because of contention for the outgoing link at Switch S_k. The worst case happens when all packets contending for the same output link at S_k arrive just before the packet of C_i. All these packets may in turn be blocked at downstream switches. The worst-case blocking delay $B_{i,k}$ at Switch S_k can therefore be defined by the following recurrence relation:

$$\begin{aligned} B_{i,k} &= \sum_{\substack{p=1\ldots n \\ p \neq I_{i,k}}} \max_{C_j \in \Delta_{p,O_{i,k},k}} \{B_{j,k-1} + e_j\} + B_{i,k-1} \\ B_{i,0} &= 0 \end{aligned} \tag{1}$$

In the above recurrence relation n is the number of ports of Switch S_k; $I_{i,k}$ and $O_{i,k}$ are the input and output port, respectively, of Connection C_i at Switch S_k; $\Delta_{p,q,k}$ denotes the set of connections that use Port p as input port and Port q as output port at Switch S_k. To simplify the notation, in the following we will generally omit the index k for $\Delta_{p,q,k}$ if it can be deducted from the context.

We say that a connection C_j *branches in* with C_i at Switch S_k if the two connections use different input ports but share the same output port at Switch S_k. Using the notation above: $I_{i,k} \neq I_{j,k}$ and $O_{i,k} = O_{j,k}$. The set of all connections branching in at with C_i at Switch S_k is denoted by $\Delta_{p \neq I_{i,k}, O_{i,k}, k}$.

Similarly, a connection is said to *branch out* from C_i at Switch S_k if the two connections share the input port but have different output ports at S_k, i.e., $I_{i,k} = I_{j,k}$, but $O_{i,k} \neq I_{j,k}$. The set of all connections branching out from C_i at S_k is denoted by $\Delta_{I_{i,k}, q \neq O_{i,k}, k}$.

Relation 1 allows for a very simple (though expensive!) centralized admission control scheme. With our simple periodic traffic model we can use the following formula to test whether packets of connection C_i can successfully be delivered before the end of the period:

$$\forall i : B_{i,K} + e_i \leq p_i \tag{2}$$

that is, whether for all connections the worst-case blocking time at their entry to the network plus the transmission time does not exceed the period.

For a new connection to be accepted, we centrally determine whether (1) the timing requirements of the new connection can be satisfied and whether (2) the timing requirements of existing connections are not violated. Since the admission of a new connection can potentially affect all previously established connections, the cost of this operation grows quadratically with the number of accepted connections. Computationally less demanding methods must be devised, in particular for systems with connections that are dynamically established and torn down.

In the following section, we describe a fully distributed admission control scheme that has a time overhead that grows linearly with the diameter of the network.

3 Admission Control Algorithm

As described earlier, packets that are blocked at a switch in a Myrinet-style network not only block other packets at that particular switch, but at upstream switches as well. Since the effect of blocked packets is not local, admission control decisions are not easily realizable locally[1]. The distributed algorithm described in this section relies on the general idea of defining *blocking guards*: for each link, a blocking guard bounds the effect connections using that link can have on other connections in the network.

Specifically, for each output Port p of a switch S_k, we define a blocking guard X_k^p, which is the longest time any packet can be blocked at that port, be it due to either existing or future connections anywhere in the network. For convenience, we define X_k^p to be zero if no connection uses Port p.

For each link on the route of a newly established connection, the admission control must make sure that (1) the worst-case blocking delay of the newly established connection does not exceed the amount specified in the blocking guard, and (2) that the new connection does not cause the worst-case blocking delay of existing connections to exceed the amount specified in the blocking guard. As connections are established and torn down, the values of the blocking guards must be modified to reflect the change in the load of the network.

In this section we describe an admission control algorithm that makes its decisions based on the values for the blocking guards and then adapts their values appropriately. In Section 4 we will describe an establishment protocol based on a distributed version of this algorithm. Before we go on to describe the algorithm in detail, we define a few terms and show a number of properties.

Let $\tilde{B}_{i,k}$ be the *calculated worst-case blocking delay* for packets of connection C_i at Switch S_k. If $BI_{i,k}$ and $BO_{i,k}$ are an upper bound on the blocking delay other connections can cause that branch in at S_k or branch out at S_k, respectively, and are defined as follows,

$$BI_{i,k} = |\Delta_{I_{i,k},O_{i,k}}| \sum_{\substack{p=1\ldots n \\ p\neq I_{i,k}}} \max_{C_j\in\Delta_{p,O_{i,k}}} \{e_j\} \tag{3}$$

$$BO_{i,k} = \sum_{\substack{q=1\ldots n \\ q\neq O_{i,k}}} \Big(\sum_{C_j\in\Delta_{I_{i,k},q}} |\Delta_{I_{i,b},O_{i,b},b}| \tilde{B}_{j,k} \Big) \ , \tag{4}$$

where S_b is the switch at which Connection C_j branches in upstream, then $\tilde{B}_{i,k}$ can be defined by the following recurrence relation:

$$\begin{aligned} \tilde{B}_{i,k} &= BI_{i,k} + BO_{i,k-1} + \tilde{B}_{i,k-1} \\ \tilde{B}_{i,0} &= 0 \ . \end{aligned} \tag{5}$$

The following theorem shows that $\tilde{B}_{i,k}$ is an upper bound for $B_{i,k}$.

Theorem 1 *For all switches S_k and all connections C_i, the following holds: $\tilde{B}_{i,k} \geq B_{i,k}$.*

[1]This is substantially simpler in store-and-forward networks, where local blocking effects cause local delays that are simply added up along the path of a connection to check against end-to-end delays.

Proof. We prove the theorem by induction over the size of the network. (In order to keep the notation manageable, we make the simplifying assumption that at most one connection branches in or branches out per port.)

Basis: The theorem holds for a network with a single switch:

$$\begin{aligned} B_{i,1} &= \sum_{\substack{p=1\ldots n \\ p\neq I_{i,1}}} e_p \\ \tilde{B}_{i,1} &= |\Delta_{I_{i,1},O_{i,1}}| \sum_{\substack{p=1\ldots n \\ p\neq I_{i,1}}} e_p, \end{aligned}$$

where C_p is the connection branching in at Port p of the switch. We see that $\tilde{B}_{i,1} \geq B_{i,1}$.

Induction step: Assuming that the theorem holds for a network with $k-1$ switches, we prove that it holds for a network with k switches as well.

In the following, n_k denotes the number of connections (C_i inclusive) that share the same input port and output port as C_i at switch S_k. If a single connection C_j branches in at S_k (and branches out at S_X), the blocking delays for C_i at S_k compare as follows:

$$\begin{aligned} B_{i,k} &= B_{i,k-1} + B_{j,X} + e_j \\ \tilde{B}_{i,k} &= \tilde{B}_{i,k-1} + n_k \tilde{B}_{j,X} + n_k e_j \quad . \end{aligned}$$

If more than one connection branches in at S_k, this argument can be repeated.

Since, by induction hypothesis, $\tilde{B}_{j,X} \geq B_{j,X}$, the theorem holds for the network with k switches as well. □

Our algorithm does not make use of $\tilde{B}_{i,k}$ directly, but of a related value, $\tilde{B}^X_{i,k}$: Let $\tilde{B}^X_{i,k}$ be the *guard-based calculated worst-case blocking delay* for packets of connection C_i at Switch S_k. If the upper bound $BO^X_{i,k}$ on $BO_{i,k}$ is defined as follows

$$BO^X_{i,k} = \sum_{\substack{q=1\ldots n \\ q\neq O_{i,k}}} \Big(\sum_{C_j \in \Delta_{I_{i,k},q}} |\Delta_{I_{i,b},O_{i,b}}| X^q_k \Big) \quad , \tag{6}$$

we can define $\tilde{B}^X_{i,k}$ by the following recurrence relation:

$$\begin{aligned} \tilde{B}^X_{i,k} &= BI_{i,k} + BO^X_{i,k-1} + \tilde{B}^X_{i,k-1} \\ \tilde{B}^X_{i,0} &= 0 \end{aligned} \tag{7}$$

The following observation shows that $\tilde{B}^X_{i,k}$ is a safe approximation for $\tilde{B}_{i,k}$, and, hence, for $B_{i,k}$. It is the basis for our algorithm.

Observation 1 *As long as for every existing connection C_j the calculated worst-case blocking delay $\tilde{B}$ at every switch S_l is bounded by the blocking guard X at the respective output port at the previous switch S_{l+1}, i.e., $\tilde{B}_{i,l} \leq X^{O_{j,l+1}}_{l+1}$, the calculated worst-case blocking delay $\tilde{B}$ of the newly established connection C_i is bounded at each switch by the guard-based calculated worst-case blocking time $\tilde{B}^X$, i.e., the following holds: $\tilde{B}_{i,k} \leq \tilde{B}^X_{i,k}$.*

The following theorem states that, at any switch, it is sufficient to check the guard-based calculated worst-case blocking delay $\tilde{B}^X$ of the new connection against the blocking guard X at the same switch to see whether any existing connection may miss its deadline.

Theorem 2 *If $\tilde{B}^X_{ik} + e_i \leq X_k^{O_{i,k}}$ holds after C_i has been established, then $B_{j,k} \leq X_k^{O_{i,k}}$ is true for every existing connection C_j that branches in at Switch S_k.*

Proof. We only need to look at the case where C_i causes the longest worst-case blocking time of all connections sharing its input port at S_k. According to Theorem 1 and Observation 1, the following inequality holds at Switch S_k:

$$\tilde{B}^X_{i,k} \geq (B_{j,k} + e_j + \Gamma) + B_{i,k-1} \quad ,$$

where Γ sums up the additional blocking time caused by the remaining connections that branch in at S_k. For every Connection C_j branching in at S_k, $B_{j,k}$ is defined as follows, according to Relation 1:

$$B_{j,k} = (B_{i,k-1} + e_i + \Gamma) + B_{j,k-1} \quad .$$

Therefore,

$$\tilde{B}^X_{i,k} + e_i - B_{j,k} \geq e_j + e_i - e_i \geq 0 \quad .$$

□

Given a set of established connections, the following three-step admission control algorithm determines whether a newly arriving connection C_i can be established along the route $S_K, \ldots, S_k, \ldots, S_1$. The algorithm rejects the new connection if (1) the performance requirements of C_i can not be satisfied, or (2) if the performance requirements of existing connections would be violated by accepting C_i. If C_i is accepted, the blocking guards along its route are updated appropriately.

Step 1: First, the guard-based worst-case blocking time $\tilde{B}^X_{i,k}$ for packets of the new connection is determined for each switch S_k along the route of C_i. If at any switch S_k the blocking guard $X_k^{O_{i,k}}$ is exceeded, that is, if $\tilde{B}^X_{i,k} + e_i > X_k^{O_{i,k}}$, establishing the new connection may violate the requirements of an existing connection, and the new connection C_i is rejected.

Step 2: If no blocking guard is exceeded, the algorithm proceeds to the *final acceptance test* at the source. The guard-based calculated blocking $\tilde{B}^X_{i,K}$ at the entrance to the network is compared to the deadline requirements, i.e., it is checked whether $\tilde{B}^X_{i,K} \leq p_i - e_i$. If this is not the case, packets of C_i may not meet their delay requirements, and the connection is rejected.

Step 3: After the new connection C_i has been accepted by the source, the blocking guards at the outgoing ports on the route of C_i must be updated to reflect the acceptance of C_i. Specifically, they may need to be reduced to prevent future connection admissions from violating C_i's performance guarantees.

If $X_{i,k}$ is the new value for $X_k^{O_{i,k}}$ after the acceptance of C_i, the following two conditions must be satisfied for the blocking guards to remain valid after the connection has been established:

Condition 1 : $X_{i,k} \leq X_k^{O_{i,k}}$: Previously established connections used the existing blocking guard during their establishment to determine a bound on their worst-case blocking time.

Condition 2 : $\tilde{B}^X_{i,k} + e_i \leq X_{i,k}$: The amount of blocking the new connection can cause for packets of other connections at S_k must not exceed the blocking guard.

The following set of formulas can be shown to generate a set of updated blocking guards that satisfy the above conditions:

$$
\begin{aligned}
X_{i,k} &:= \min(X_k^{O_{i,k}}, X_{i,k+1} - BI_{i,k+1} - BO_{i,k}^X - R_{i,k+1}) \\
R_{i,k} &:= \frac{X_{i,k} - \tilde{B}_{i,k}^X - e_i}{k}
\end{aligned}
\qquad
\begin{aligned}
X_{i,K+1} &:= p_i \\
R_{i,K+1} &:= 0
\end{aligned}
\tag{8}
$$

where BI and BO^X have been defined earlier.

Intuitively, the term $R_{i,k}$ gives an indication of how much additional blocking can be accepted at output Port $O_{i,k-1}$ in the future, without violating the blocking guard $X_k^{O_{i,k}}$ at S_k. We call $R_{i,k}$ the *local residual time* between the two ports. Updating the blocking guards means allocating residual time to links along the route of the new connection.

The above formula trivially satisfies Condition 1. The following theorem shows that the formula satisfies Condition 2 as well.

Theorem 3 $X_{i,k} \geq \tilde{B}_{i,k}^X + e_i$.

Proof. We need to distinguish two cases:

Case 1: $X_k^{O_{i,k}} < X_{i,k+1} - BI_{i,k+1} - BO_{i,k}^X - R_{i,k+1}$
The theorem trivially holds in this case, since Connection C_i would have been rejected earlier otherwise.

Case 2: $X_k^{O_{i,k}} \geq X_{i,k+1} - BI_{i,k+1} - BO_{i,k}^X - R_{i,k+1}$
In this case, the theorem is equivalent to

$$X_{i,k+1} - BI_{i,k+1} - BO_{i,k}^X - R_{i,k+1} \geq \tilde{B}_{i,k}^X + e_i \quad . \tag{9}$$

This can be rewritten to

$$X_{i,k+1} - R_{i,k+1} \geq \tilde{B}_{i,k+1}^X + e_i \quad , \tag{10}$$

which is equivalent to

$$R_{i,k+1} \leq X_{i,k+1} - B_{i,k+1} - e_i \quad , \tag{11}$$

which is true by construction of $R_{i,k+1}$. □

The correctness of the admission control algorithm follows directly from the above theorems: On one side, it never accepts a connection whose packets may miss their deadlines. This is enforced by the acceptance test at the source. On the other side, the admission control algorithm never accepts a connection if this may cause packets of existing connections to miss their deadlines. This is enforced by comparing an upper bound on the blocking delay ($B_{i,k}^X$) with blocking guards at the intermediate switches. Theorem 2 states that this test is sufficient. Theorem 3 shows that the algorithm updates the blocking guards correctly to reflect the acceptance of C_i.

4 The Distributed Admission Control Protocol

The admission control algorithm described in Section 3 lends itself as basis for a distributed connection establishment protocol for real-time connections over wormhole networks.

The protocol consists of a set of *host establishment controllers* running on the hosts connected to the network, and a set of local *switch establishment controllers* (SECs), one per switch. The

SEC can reside in the switch controller itself (this solution is not possible in Myrinet switches,) or on a host (or the network interface controller of that host) connected to the switch, or even on a remote host (or its network interface controller).

The SEC of a switch maintains the blocking guards at that switch and other information about the currently established connections that traverse the switch, or connections that are in the process of being established.

4.1 Connection Establishment

An application seeking to establish a connection contacts the local HEC, which, in turn initiates a *three-pass connection establishment procedure*. (Although we limit ourselves to a source-initiated, source-routed scenario, we will describe later how we plan to take advantage of the fact that this scheme can be adapted to destination-initiated or adaptive routing schemes as well.)

Pass 1 The HEC at the sender host sends an *establish request* message, which travels from SEC to SEC of switches along the connection route to the destination. The *establish request* message for the new connection C_i contains the worst-case transmission time e_i for packets of C_i and information about connections that branch in at switches upstream on the route of C_i. Specifically, the *establish request* message for the new connection C_i contains the following entries:

- e_i : The worst-case message length of Connection C_i.
- $\mathcal{BI}$: A list of entries of the form (C_j, n_j), that specifies all connections that branch in at switches upstream on the route of C_i to the sender host. C_j specifies the connection identifier of the branching-in connection. The value of n_j indicates the number of connections (including C_i) that share the same input *and* output port as C_i at the switch where C_j branches in. In other words, n_j is equal to $|\Delta_{I_{i,k},O_{i,k}}|$ if C_j branches in at S_k.

Upon receiving an *establish request* message for connection C_i, the local SEC at Switch S_k performs the following operations before forwarding the message to the next SEC on the route of C_i:

1. For every connection C_p that branches in at S_k, append the entry $(C_p, |\Delta_{I_{i,k},O_{i,k}}|)$ to the list $\mathcal{BI}$ of the *establish request* message.
2. For every connection C_o that branches out at S_k, look up the corresponding entry (C_o, n_o) in the list $\mathcal{BI}$ of the *establish request* message, and remember n_o. Delete the entry from the list $\mathcal{BI}$.
3. Remember e_i.

Pass 2 When the *establish request* message reaches the destination host, the second pass is invoked by sending an *establish reply* message upstream to the sender host. As the message is sent from S_{k-1} to S_k, it contains a single entry:

- B : The guard-based calculated worst-case blocking time at S_k for packets of C_i, assuming that no connections branch in at S_k. Formally, $B = BO^X_{i,k-1} + \tilde{B}^X_{i,k-1}$, using BO^X as defined in Section 3. (The value of B is zero when the message is sent from the destination host to the next switch upstream.)

Upon receiving an *establish reply* message for C_i from S_{k-1}, the local SEC at S_k determines the local value for $BI_{i,k}$, and can now determine the guard-based worst-case blocking delay $\tilde{B}^X_{i,k}$. It then checks this value against the local blocking guard. Specifically, the SEC performs the following operations:

1. Determine the guard-based calculated worst-case blocking delay $\tilde{B}^X_{i,k}$ for C_i at this switch by adding the blocking effect of connections that are branching in to B:

   $$BI_{i,k} + B = BI_{i,k} + BO^X_{i,k-1} + \tilde{B}^X_{i,k-1} = \tilde{B}^X_{i,k} \ . \tag{12}$$

   In this case, $BI_{i,k}$ can be determined locally, and B is passed in the *establish reply* message. Also, remember the old blocking guard value for use during teardown.

2. Check whether the global worst-case blocking delay is bounded by the local blocking guard, i.e., whether $\tilde{B}^X_{i,k} + e_i \leq X_k^{O_{i,k}}$. If the inequality does not hold, reject the connection and abort the establishment process. Otherwise, continue; remember $\tilde{B}^X_{i,k}$.

3. Update the entry B of the *establish reply* message to be $B = \tilde{B}^X_{i,k} + BO^X_{i,k}$, where $\tilde{B}^X_{i,k}$ was calculated earlier, and $BO^X_{i,k}$ can be locally calculated with help of the local blocking guards and the n_o values stored in Pass 1. Forward the updated value to the next SEC upstream along the route of C_i to the sender.

Pass 3 When the *establish reply* message reaches the HEC at the source host, the *final acceptance test* is performed by comparing the value in the establish reply with the deadline requirements of C_i. If the test fails, i.e., the inequality $B + e_i \leq p_i$ does not hold, the connection is rejected and the establishment procedure aborted. Otherwise, the third and final pass is invoked by sending an *establish confirm* message along the route of C_i to the destination host. When sent from S_{k+1} to S_k, this message contains the following entries:

- $X = X_{k+1}^{O_{i,k+1}} - BI_{i,k+1} - R_{i,k+1}$.
- k : The number of remaining hops to the destination host.

The initial value of X at the HEC is D_i, and k is the total number of hops in C_i's path. Upon receiving an *establish confirm* message, the SEC at S_k first determines the new value for the local blocking guard, then determines the new residual time, and finally forwards the message with the updated entries X and k. In detail, the SEC at S_k performs the following operations:

1. Update the blocking guard $X_k^{O_{i,k}}$ to be $X_{i,k}$ as follows:

   $$\begin{aligned} X'_{i,k} &:= X - BO^X_{i,k} \\ X_{i,k} &:= \min(X_k^{O_{i,k}}, X'_{i,k}) \ , \end{aligned}$$

   where X is passed with the *establish confirm* message, and $BO^X_{i,k}$ was locally calculated for Pass 2.

2. Determine the new value for the proposed residual time $R_{i,k}$ between this and the next switch as follows:

   $$R_{i,k} := \frac{X_{i,k} - \tilde{B}^X_{i,k} - e_i}{k} \ ,$$

   where e_i and $\tilde{B}^X_{i,k}$ were stored locally during the earlier passes.

3. Append C_i to the local list of established connections and remember $X'_{i,k}$

4. Forward the *establish confirm* message to S_{k-1} with the updated entries $X_{i,k} - BI_{i,k} - R_{i,k}$ and $k-1$.

When the *establish confirm* message reaches the SEC at the receiver host, the establishment process is completed.

4.2 Connection Teardown

In principle, the *connection teardown procedure* is simple: A *teardown request* message is sent along the path of the connection and instructs all SECs to update their lists and recalculate (that is, increment) their blocking guards.

Unfortunately, a connection C_j admitted after C_i may depend on the blocking guard previously allocated (that is, reduced) when C_i was established. Simply increasing the value for the blocking guard when C_i is torn down may cause future connection establishments to violate C_j's delay requirements. The update of the blocking guard after the teardown of C_i must therefore be delayed until no connection remains that relies on the blocking guard allocated during C_i's establishment. We say that Connection C_i is *critical* for another connection C_j at Switch S_k if C_j relies on the blocking guard as computed after the establishment of C_i. More formally, C_i is critical for C_j if the following conditions hold:

1. C_j was established after C_i.

2. C_j branches out (from the route of C_i) at S_k.

3. $X_k^{O_{i,k}} = X'_{i,k}$ when C_j was established. (Note that C_i may have been torn down in the meantime.)

After Connection C_i is torn down, the update of the blocking guard on S_k is delayed until all connections for which C_i is critical are torn down.

The establishment and teardown protocols maintain the criticality relation for each switch in the form of an adjacency list by storing two lists, $\mathcal{A}_{i,k}$ and $\mathcal{B}_{i,k}$, for each connection C_i, where $\mathcal{A}_{i,k}$ is the set of connections for which C_i is critical on S_k, and $B_{i,k}$ is the set of connections that are critical to C_i.

When a connection C_j is established at Switch S_k, the entries for $\mathcal{A}_{j,k}$ and $\mathcal{B}_{j,k}$ are created, and the adjacency lists at that switch are updated. When a connection C_i is torn down at Switch S_k (that is, the *teardown request* message reaches the SEC at Switch S_k), the following sequence of operations is performed:

1. Remove C_i from the list of accepted connections.

2. Check if $\mathcal{A}_{i,k}$ is empty. If so,

   (a) If $X_k^{O_{i,k}} = X'_{i,k}$, update $X_k^{O_{i,k}}$: $X_k^{O_{i,k}} := \min(X'_{l,k})$ for all connections C_l that share the same output port as C_i.

   (b) Otherwise, do not update $X_k^{O_{i,k}}$.

   (c) Delete $\mathcal{A}_i$.

3. For each $C_j \in \mathcal{B}_{i,k}$, delete C_i from $\mathcal{A}_{j,k}$. If $\mathcal{A}_{j,k}$ becomes empty, proceed as described in the step above.

Once the *teardown request* message reaches the destination host, the teardown procedure is complete. By following the above described procedure, we ensure that the calculations carried out in determining the *worst case blocking time* of connections established after C_i are still valid. This correspondingly ensures that the tearing down of C_i does not disturb any connection sharing a link with C_i, whose Quality of Service requirements were dependent on the *maximum* amount of blocking limit guaranteed at that switch by C_i.

5 Experimental Results

A suite of simulation experiments were performed to determine the performance of the proposed establishment protocol. This allowed us to analyze the behavior of the protocol for different network topologies, varying network loads and bandwidth requirements of the connections. We assumed two different network topologies, a 4x4 Mesh with 16 four-port switches and 12 hosts and a 6x6 Torus with 36 five-port switches and 36 hosts.

For the simulations we assumed exponential distributions for both connection interarrival times (with λ being the arrival rate), and connection durations (with μ being the service rate).

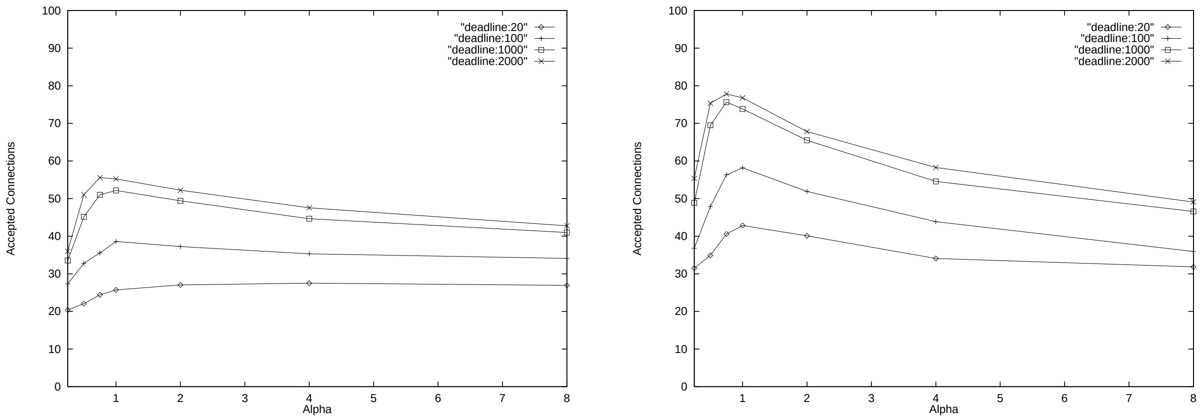
For each of the experiments, we assumed a network topology and increased the load on the network by repeatedly generating random connections which arrive in the network at a rate of λ, trying to establish them, and if they do get admitted into the network, tear them down at a rate of μ. (For simplicity, we restricted ourselves to connections with identical traffic parameters.) As our criteria for evaluating the performance, we measured the average number of currently admitted connections in the network, during steady state. A related measure is the average link utilization, which shows how much the accepted connections utilize the existing links.[2]

As a reference for comparison, we use the centralized admission control described in Section 2. We recall that the centralized algorithm is an optimal admission control test, in that, given a set of connections, the test fails if and only if Recurrence Relation (1) is violated for at least one of the connections. Both the distributed and centralized algorithms operate on-line, that is, without any knowledge of future connection arrivals. Since we are not addressing routing in this work, we use the simplest possible routing algorithm, in our case X-Y routing. With a confidence level of 95%, the confidence intervals for all depicted results are below 2% of their absolute value. The interval at which we sample the number of connections admitted in the network is chosen sufficiently large to eliminate correlation effects.

Conditions 1 and 2 used to determine the residual time leave some room in recalculating the blocking guards. Formula (8) attempts to distribute the residual time equally over the route of the connection. This basic allocation formula can be modified to vary the allocation of residual time, such as allocating more residual time toward the sender (causing lower values for the blocking guards) or toward the receiver (resulting in higher blocking guards). Two effects are traded off when updating the value for the blocking guard. On one hand, a large value for the blocking guard gives more room to accept connections downstream from the switch, whereas smaller blocking guards tend make it more difficult to accept connections downstream. On the other hand, future connections *upstream* are penalized by a larger blocking guard, since the latter causes higher values for guard-based calculated worst-case blocking times $\tilde{B}^X$.

Figure 2 illustrates the effect of varying the allocation of residual time on the number of accepted connections for a fixed load. In this series of experiments, we attempted to allocate the residual time in the form $R, \alpha R, \alpha^2 R, \ldots, \alpha^K R$, and plotted the performance for varying α's. The

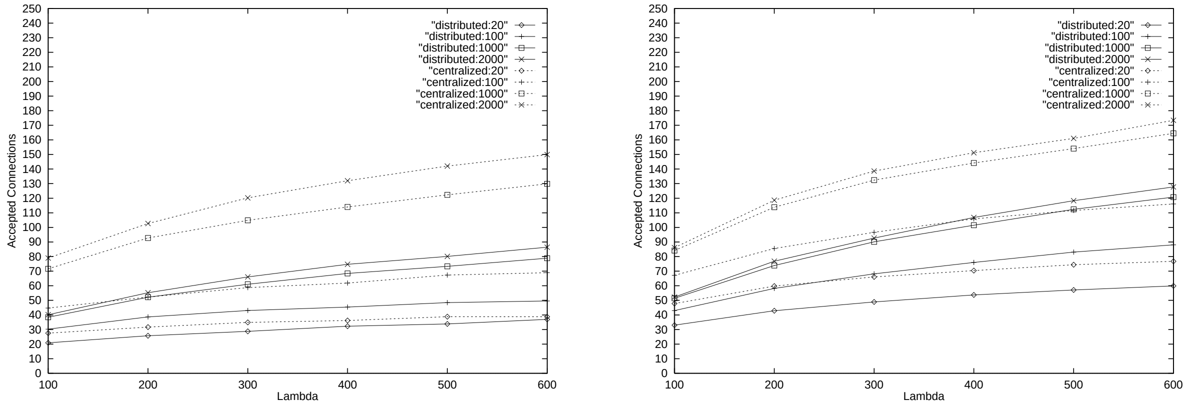
[2] An approximation of the latter measure can be easily derived from the number of admitted connections.



(a) Mesh Topology (b) Torus Topology

Figure 2: Experimental Results for varying α and $\lambda = 200$

results are collected for an interarrival rate of $\lambda = 200$ and different bandwidth requirements of connections. This indicates that overall, the best results in terms of number of connections accepted are obtained with for a value of α close to 1.0, which corresponds to the equal distribution approach defined in Formula (8). We will thus focus our attention on data generated using Formula (8) for the remaining analyses.



(a) Mesh topology (b) Torus topology

Figure 3: Experimental Results on a 4x4 Mesh and 6x6 Torus

Figure 3 shows the experimental results for the 4x4 Mesh. The results are for worst-case message lengths e_i of one time unit and varying interarrival times p_i of 20, 100, 1000 and 2000 units. For example the case of $e_i = 1$ and $p_i = 1000$ corresponds to the requirements of uncompressed CD audio over 640 Mb/s links. As to be expected, the number of accepted connections increases with increasing arrival rate. Also, obvious is the fact the high-bandwidth connections show lower acceptance probabilities, owing to the fact that they generate higher blocking delays. We see that the centralized admission control scheme performs better. Having access to a global state of the network at all times enables the centralized scheme to optimally allocate resources and admit a larger number of connections.

To compare the performance of the algorithm for different network topologies, we performed a similar set of experiments on a 6x6 Torus topology, also shown in Figure 3. This topology has a higher host-per-link ratio and thus less contention for links among connections. This is reflected in the higher numbers for accepted connections.

From the above graphs, we see that although the *number* of accepted connections *increases* with λ, the *rate* of acceptance *decreases*. Also evident is the fact that for the same λ value, lower bandwidth requirements yield in higher number of connections being accepted. This is verified in the Mesh topology for a λ value of 300 and a bandwidth requirement of 1/2000, where the percentage of accepted connections is 66/300 = 22%. Maintaining the same arrival rate, we observe that if the bandwidth requirement is doubled to 2/2000, the percentage of accepted connections drops to 61/300 = 20.33%, which is not a significant drop. If instead of doubling the bandwidth requirement, we double the arrival rate to 600, the percentage of accepted connections drops significantly to 86.45/600 = 14.4%. This illustrates the well known fact that in wormhole networks, the blocking delays are substantially more sensitive to an increase of the number of connections than of the bandwidth requirements per connection.

6 Summary and Conclusion

Wormhole routing is a cost-effective way to provide very-high speed, very-low latency communication for emerging distributed real-time applications with high-bandwidth demands. Unfortunately, the lack of traffic buffering in the switches makes it difficult to give real-time guarantees to traffic streams that contend for communication links.

In this paper we propose a simple blocking model for traffic on wormhole-routed networks and derive a fully distributed admission control scheme for periodic traffic streams. The scheme is based on a three-pass message exchange among the switches on the connection route. Experimental comparisons of a straight forward implementation of our scheme with a fully centralized algorithm are encouraging. By providing tighter bounds in the calculation of our calculated worst-case blocking times and appropriately modifying the protocol, we can improve the performance of the distributed algorithm as compared to the centralized scheme.

Rate control at the sender host [5] is known to significantly improve the real-time performance of wormhole networks. We are currently studying how to use our current distributed algorithm as a framework to include and make use of information about rate control at the sender.

This distributed scheme has a number of non-obvious benefits. First, it can be seamlessly integrated into existing admission control schemes for store-and-forward networks (e.g. [9, 11]), and thus allow a well-integrated real-time service over hybrid (that is, store-and-forward and wormhole-routed) networks. Second, guaranteeing timing requirements subsumes deadlock avoidance. Our scheme can therefore support a distributed, adaptive routing scheme, which in turn rely on our scheme to guarantee deadlock freedom. Since this study is strictly about admission control, we are currently not taking advantage of this, and establish the connection with a *a priori* route. This is not required by our protocol, however, and we are currently investigating adaptive, load-sensitive routing schemes.

During connection establishment, resources can be allocated without involving nodes that are not on the connection route. We can also incorporate resource deallocation done by network management. This periodically discovers allocated, but unused resources (due to connections torn down), and makes them available to new connections.

The only assumption we make about the switches in this work is that they serve waiting packets in a round-robin manner. There are current efforts (e.g. [10]) to study the benefits of special support for real-time traffic in wormhole-routed switches. We are currently investigating how such additional functionality (e.g. priorities for packets) affects the performance of our scheme.

Substantial work is currently being done on host and switch supported multicasting over wormhole networks (e.g., [8]). The problem of how to guarantee real-time constraints in such a setting still needs to be addressed.

References

[1] G. Agrawal, B. Chen, W. Zhao, and S. Davari, Guaranteeing Synchronous Message Deadlines with the Timed Token Protocol. In *IEEE Computer*, Vol. 43. No. 3, March 1994.

[2] A. Banerjea and S. Keshav. Queueing delays in Rate Controled ATM Networks. In *Proceedings of INFOCOM '93*, San Francisco, March-April 1993.

[3] Riccardo Bettati and Anisoara Nica. Real-Time Networking over HIPPI. In *Proceedings of the Workshop on Parallel and Distributed Real-Time Systems*, Santa Barbara, California, April 24-26, 1995.

[4] Riccardo Bettati, Domenico Ferrari, Amit Gupta, Wendy Heffner, Winnie Howe, Mark Moran, Quyen Nguyen and Raj Yavatkar. Connection Establishment for Multi-Party Real-Time Communication. In *Proceedings of the 5th International Workshop on Network and Operating System Support for Digital Audio and Video*, Durham, NH, April 1995.

[5] B. Chen, H. Li and W. Zhao. Meeting Delay Requirements in Computer Networks with Wormhole Routing. In *Proceedings of the IEEE International Conference on Distributed Computing*, Hong Kong, May 1996.

[6] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C.L. Seitz, J. N. Seizovic and W.-K. Su. Myrinet – A Gigabit-per-Second Local-Area Network. In *IEEE-Micro*, Vol.15, No. 1, pp. 29-36, February 1995.

[7] R. Felderman, A. DeSchon, D. Cohen and F. Finn. ATOMIC: A High Speed Local Communication Architecture. In *Journal of High Speed Networks*, Vol. 3, No. 1., 1994, pp. 1-30.

[8] M. Gerla, P. Palnati and S. Walton, "Multicasting Protocols for High-Speed Wormhole-Routing Local Area Networks," in *Proceedings of the ACM SIGCOMM'96*, Stanford, CA, August 1996.

[9] Domenico Ferrari, Anindo Banerjea and Hui Zhang. Network Support for Multimedia: A Discussion of the Tenet Approach. In *Computer Network and ISDN Systems*, pages 1267–1280, July 1994.

[10] J.-P. Li and M. W. Mutka. Priority Based Real-Time Communication for Large Scale Wormhole Networks. In *Proceedings of the IEEE IPPS '94, 8th International Parallel Processing Symposium*, pp. 433-438, April, 1994.

[11] Craig Partridge and Stephen Pink. An Implementation of the Revised Internet Stream Protocol (ST-2). In *Journal of Internetworking Research and Experience*, pages 27-54, 1992.

[12] Lixia Zhang, Steve Deering, Deborah Estrin, Scott Shenker and Daniel Zappala. RSVP: A New Resource Reservation Protocol. In *IEEE Networks Magazine*, Vol. 31, No. 9, pages 8-18, September 1993.