12-31-1991

Communications network for distributed real-time systems

Charles Silva
New Jersey Institute of Technology

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

Communications Network For Distributed Real-Time Systems

Charles Silva
Real-Time Computing Laboratory
Department of Computer Information Science
New Jersey Institute of Technology
Newark, NJ 07102
U.S.A.

Master's Thesis

Abstract

The nature of predictable real-time systems requires that the communications network with which these systems operate, needs to also be predictable. To make this system entirely predictable, the time it takes to transmit data on the network from one station to another, must be known. Most of today's network architectures and protocols do not provide communications in a predictable fashion. This paper will present a communications architecture and protocol using a multiple token ring network.

Approvals

Date Submitted: 12/23/91

Date Approved: 12/23/91

Approved By (Faculty Advisor): _____

Table of Contents

1 Introduction

Real-time communication networks differ from conventional networks in that the former are constrained by time. They need to insure timely delivery of messages between nodes on the network. Due to this, performance of real-time networks is measured differently from conventional networks.

The primary performance measure for real-time networks as opposed to conventional networks is arrival time of messages at their destinations. For conventional networks, the primary performance considerations are to maximize the network throughput and to minimize the average delay. However, for real-time networks, we need to insure that message arrival time at a destination will be predictable. In order to do this, we need to start by designing a network architecture and protocol that will provide predictable communications.

Many communications architectures, such as the IEEE 802.3 CSMA/CD standard, do not provide predictable performance. As offered load increases, so does throughput until, beyond its maximum value, throughput declines as load increases[1]. With token ring networks, it has been studied that these networks have good properties of bounded transfer delay. However, a single ring network, at times of heavy load, introduces delay times which although predictable, are too long for real-time systems. It is for this reason that a multiple token ring network lends itself well to real-time systems.

Multiple token ring networks are different from conventional token ring networks in that the former use multiple token rings connected via bridges. When a node on the network wishes to send a message to a node which is not located on the same ring, the message is routed over a bridge to the destination ring. Each ring on the network maintains its own token. By dividing the number of nodes onto separate rings, the time it takes for a node to receive the token is significantly reduced. This is especially important in the area of predictable real-time systems.

Using a multiple token ring network, average waiting time for a token is reduced, yet since the upper bound arrival time can be calculated based on the network architecture, the network upholds the requirement of predictability.

Also, by using a multiple token ring network, the traffic incurred by the network is divided up over the different rings. In this case, the frequency with which a station receives the token is increased. Since the traffic is reduced, the number of

messages which arrive before their deadline arrival time increases. By providing this performance, this network becomes a suitable device for providing network performance.

Since performance of multiple token ring networks is improved by dividing the number of stations onto multiple rings and it provides predictable performance, it has been concluded that this network would provide a distributed real-time system with considerable performance and predictable communications. This architecture, discussed later in full detail, provides reliable performance while maintaining necessary predictability. This paper will review previous research and simulations done in this area, present the design and implementation, and discuss simulation results acquired from simulations done with different network performance measures. In section one, this paper will start by briefly summarizing a group of articles and books written in the area of real-time systems and time-constrained communications.

In section 2, this paper will present the design of the architecture of this network. Topics covered will consist of network architecture, network topology, network protocol, packet layout, and bridge design.

In section 3, this paper will present the operation of the network as performed by a simulation. Covered topics include simulation initialization, simulation operation, object to station communications, calculation of station delay time, delivery of messages, and token passing.

In section 4, the network reporting mechanism will be covered. This section will cover the reporting mechanisms used for each station and ring, and discuss the reporting mechanism for the entire network.

In section 5, module design will be presented. This will consist of describing in detail each module which makes up the system, and describing which functions the modules perform.

In section 6, system evaluation and results will be presented. These results will consist of qualitative and quantitative results obtained from performing simulations based on different network architectures.

Finally in section 7, future work to be done in this area will be discussed.

1.1 Previous Work

In the article written by Shin and Hou[6], the authors analyze three protocols for use with real-time systems. The three protocols consist of the token passing protocol, the token ring scheduling protocol, and the P_i-persistent protocol. They begin by defining the performance parameters of *probability of dynamic failure*, P_{dyn}, and *e-bounded delivery time*, T_e.

Probability of dynamic failure is defined as the probability of a station failing to deliver messages before their deadlines. In a real-time environment, it is necessary that a message arrives at its destination before its deadline delivery time. If it does not, it becomes irrelevant data.

e-Bounded delivery time is defined as the time period between the arrival of a packet and its delivery with probability greater then $1-e$**[6].**

Using these two measures, performances of the three protocols listed above, are evaluated and compared over a wide range of traffic. In the **token passing protocol**, a station which has possession of the token can transmit data over the network until it returns. Upon receiving the returning token the station passes it on to the next station. In **token ring scheduling protocol**, each station examines the reservation field as the token passes and inserts the priority of its packet if and only if its priority is higher than the one currently in the reservation field**[6]**. A station is allowed to capture the token and transmit its packet only when the token returns with the station's claimed priority after passing through other stations. Thus, the token has to come to the station twice before it can transmit its packet. In the **P_i-persistent protocol**, each station monitors the channel constantly, and persists to transmit its packet in a slot with the probability of p_i. When there is a collision the station again persists to transmit with the probability of p_i until the transmission is complete. Protocols for use with real-time systems are also discussed in Lee and Shen[1], Tanenbaum[2], Magee and Kramer[11], Schwartz[5], Kurose and Schwartz[22], and Strosnider, Marchok, and Lehoczky[14].

In the paper written by Lee and Shen**[1]**, the authors propose a multiple channel, multiple token ring architecture. Instead of examining the conventional performance requirements for networks, the authors base there architecture on real-time system requirements. The proposed architecture and protocol contain the following advantages**[1]**:

- It is a dynamic policy which is flexible to various application situations, it can also be applied to conventional communications as well as time-constrained communications.
- It can minimize the lost percentage of critical message packets while maintaining a high channel utilization.
- It is easy for the implementation by possibly expanding the existing ring interface technologies.

In their proposal, a control protocol, serves as a distributed scheduling mechanism by imposing an implicit or explicit transmission order on the packets distributed among the stations in the ring along with serving as an arbiter of channel accessing and sharing**[1]**.

By performing simulations, Lee and Shen show that this architecture minimizes the number of lost critical messages while maintaining a high channel utilization. Architectures for real-time system networks are also presented in Strosnider and Marchok[7], Chen and Bhuyan[8], Martin[11], Magee and Kramer[12], Hayes[15], and Lee, Schaff, Tsai, and Srinivasan[15].

In the paper written by Leung, Tam, and Young**[28]**, the authors study the problem of routing equal-length messages in several networks: **unidirectional ring, out-tree, in-tree, bidirectional tree and bidirectional ring.** Also the authors consider the issue of whether it is possible to have an optimal on-line routing algorithm under various restrictions of four parameters: origin node, destination node, release time and deadline. For a unidirectional ring, the authors show that no optimal on-line algorithm can exist unless one of the four parameters is fixed. **For out-tree**, they show that no optimal on-line algorithm can exist unless one of the following three is fixed: origin node, destination node, and release time. For **in-tree**, the authors show that no optimal on-line algorithm can exist unless origin node, destination node, or deadline is fixed. For the **bidirectional tree**, they show that no optimal on-line algorithm can exist unless either the origin node or the destination node is fixed. Finally, for the **bidirectional ring**, the authors show that no optimal on-line algorithm can exist unless the origin node and either the destination node or the release time are fixed. Other research done in the area of message routing in networks is discussed in Dixon and Pitt[26], Choi[25], Shin and Hou[6], Chen and Bhuyan[8], Magee and Kramer[12], Chlamtac and Ganz[21], and Kurose, and Schwartz[22].

In the article by Dixon and Pitt[**26**], the authors present methods for addressing, routing, and use of bridges in token ring networks. First the authors define the basics of token ring networks and the two types of token. The first, a **free token** indicates the right to transmit. When a station receives a free token and has data units to transmit, it changes the configuration of the token to that of a **busy token** and includes the busy token as part of each data unit it then transmits. The data units travel from station to station around the ring. Each station that receives a data unit checks the address in the data unit to see if it should process that data unit. In either case, it sends the data unit to the next station. Next, the authors describe the routing and use of bridges in multiple token ring networks.

Bridges used in a multiple token ring network are implemented in two different ways. The *transparent bridge* accepts every transmitted data unit. It then looks up the destination address in a hash table. From this it can determine whether or not to transmit the data unit onto the connecting ring. In the *source routing bridge* method, the source station knows the path the data unit needs to take in order to reach its destination. When the source station builds the packet, it also includes the path of bridges to take when sending the data unit. When a bridge receives this data unit, it analyzes the path and determines if it needs to transmit the data unit on the connecting ring. The study of bridges is also covered in Lee and Shen[1], Tanenbaum[2], Backes[3], Strosnider and Marchok[7], Chen and Bhuyan[8], Hayes[15], Chlamtac and Ganz[21], and Hamner and Samsen[27].

In the book written by Schwartz[**5**], the author defines performance parameters relative to ring networks. All messages (frames) move around the ring and are actively repeated by each station through which they pass. A station reading its own address as a destination copies the frame, while passing it on, to the next station on the ring. The information contained in the frame will then be passed on to other devices connected to the station. A circulating frame is removed from the ring by the station that transmitted it. The time required to repeat the frame at a station, i.e., the delay through the station, is termed the *latency* at the station. The ring latency is the sum of the propagation delay $\mathbf{t}^1$ around the ring plus the latency at each station. Thus, the ring latency is defined as:

$$\mathbf{L = t + Nb/C}$$

---

1 These parameters are expressed in quantums of time as they relate to the simulation time.

where $\mathbf{t}^1$ is the propagation delay around the ring, N is the number of stations actively connected to the ring, $\mathbf{b}^1$ is the station latency, and C is the transmission capacity. Ring performance parameters are also presented in Tanenbaum[2], Stallings[3], Chen and Bhuyan[8], Macnair[10], Sethi[13], Hayes[15], Bux[20], Chlamtac and Ganz[21], Bertsekas and Gallager[24], and Pitt[29].

2 System Design

The main purpose for this system is to provide predictable communications for a distributed real-time system simulation. The network therefore, must be designed in such a way that the time it takes for one station to send a message to another station can be calculated. To satisfy these requirements, the network architecture and protocol are based on a multiple token ring network. In this section, the network architecture, protocol and bridge design will be presented.

2.1 Network Architecture

2.1.1 Network Topology

The architecture for this network is a multiple token ring network. Each of the rings maintains its own token to allow communication within the ring. To communicate to stations on another ring, a bridge is used.

Bridges, along with stations, are placed on a ring based upon a topology file which is read in at simulation initialization. The file contains the types of nodes which can appear on the network. Each number represents a type of node. The definitions for these nodes are listed here:

- 1 - CU : The **CU** or *Control Unit*, provides a means of controlling the processing of instructions.
- 2 - ALU : The **ALU** or *Arithmetic Logic Unit*, executes instructions. It is so called in recognition of the fact that instructions either involve numerical operations (arithmetic) or non-numerical (logical) operations, such as program branching and symbolic processing.
- 3 - Register : *Registers* are those components which provide a storage device for words.
- 4 - PSW : A **PSW** or *Program Status Word*, stored in a special register, indicates program status, interrupts that the CPU may respond to, and the address of the next instruction to be executed.
- 5 - IOP : An **IOP** or *Input-Output Processor* is a special-purpose processor used exclusively to control input-output operations.
- 6 - Main Mem. : *Main Memory* is a large fast memory used for program and data storage during computer operation. It is characterized by the fact that location in main memory can be accessed directly and rapidly by the CPU instruction set.
- 7 - Secondary Mem. : *Secondary Memory* also called auxiliary memory, is generally much larger in capacity but slower than main memory. It is used for storing files which

are not continually accessed by the CPU; it also serves as an overflow memory when the capacity of main memory is exceeded.

- 8 - Command Processor : A *Command Processor* reads an instruction, and based on what is to be performed, distributes the work to the respective resources.
- 9 - OS Kernel : The **OS** or *Operating System Kernel* is a portion of the operating system that remains continuously in main memory and consists of the most frequently used part of the operating system.
- 0 - End Of Ring
- -# - Bridge

The order in which these numbers are listed in the file represents the topology of the network. A negative number represents the number of a bridge on a ring and a 0 represents the end of a ring.

An example of this file is displayed in Figure 2-1.

```
1
6
3
-1
2
0
5
7
-1
4
-2
0
8
9
2
-2
-3
0
-3
1
2
6
9
4
0
```

Figure 2-1: Sample network topology file.

For the figure above, the first ring contains a CU, Main Memory, a Register, a Bridge, and an ALU. A complete layout is displayed in Figure 2-2. In this sample

file displayed in Figure 2-1, the first five numbers correspond to items on the first ring. The sixth number, a zero is the ring delimiter. The first five numbers are a one, a six, a three, a negative one and a two. Therefore, by translating these numbers according to the list above, the first item on the ring is a CU, the second is Main Memory, the third is a Register, the fourth is a Bridge, and the fifth item on the ring is a ALU.

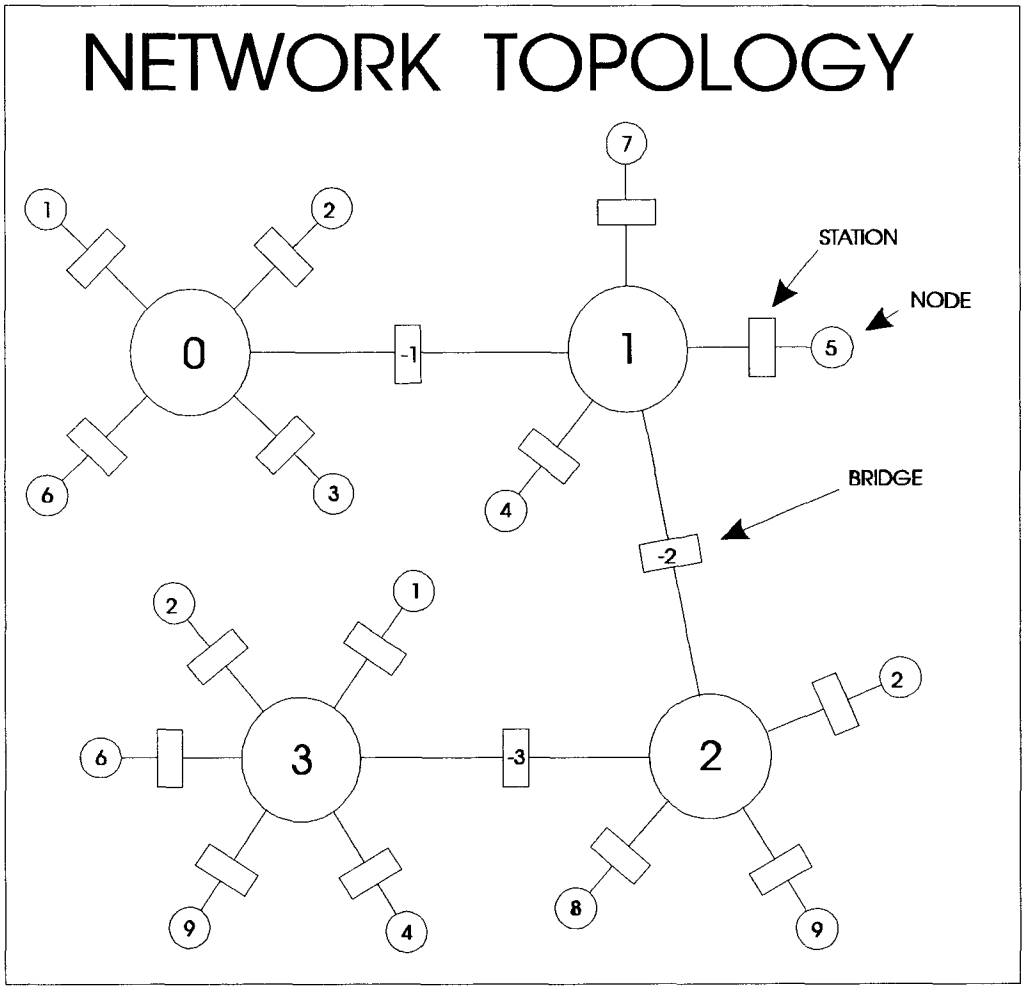

Figure 2-2: Network topology based on Figure 2-1.

This topology is defined statically at the start of the simulation. For tests of different topologies, this file can be changed for each simulation.

2.1.2 Routing Table

Another file which is read in at simulation initialization is the ring routing table. This table provides the network with the information necessary for it to transmit messages over multiple rings. For each ring on the network, this file will tell the network which bridge it must use to reach a destination on another ring. The data provided in the file is the destination ring and the local bridge needed to reach that ring. A sample of this file corresponding to the network topology described in Figure 2-1 is displayed in Figure 2-3.

Each set of routing information is divided by rings and is delimited by a -99. In this example, when a station on ring number 1 wishes to send data to a station on ring number 2, the network will know to route the message to bridge number -2 on its own ring in order to reach its destination. This information is especially useful when a message needs to cross over multiple rings to reach its destination. For example, when a station on ring number 0 needs to send a message to a station on ring number 3, the network need not find an absolute path from one station to the other over all the rings located on the network. The network needs only to analyze the destination address, determine which ring it is located on, and look up in the network routing table, which bridge to use. The network can then route the message to that bridge, and when the bridge receives the token on the next ring, the same procedure is repeated.

```
1      -1
2      -1
3      -1
-99     0
0      -1
2      -2
3      -2
-99     0
0      -2
1      -2
3      -3
-99     0
0      -3
1      -3
2      -3
-99     0
```

Figure 2-3: Routing table based on Figure 2-1.

2.1.3 Performance Measures

At simulation initialization, certain performance measures must be defined. These measures are defined to the simulation in order to calculate token and packet arrival time at a station. The measures which are required by the software are:

- Node Delay
- Node to Node Transmission Time
- Ring Latency

These can be varied from simulation to simulation to measure how these variables affect the operation of the real-time system. These variables are expressed in quantums of time as they relate to the simulation. With these measures, it is possible to calculate when something will happen on the network. With the ability to calculate occurrences on the network, it is demonstrated that this network provides predictable performance.

By having the network layout defined at the beginning of the simulation, the upper-bound time it would take to transmit a message from one station to another can be calculated. Based on this, the number of messages arriving before and after deadline time can be calculated.

2.2 Network Protocol

The protocol used in this network is based simply on that of normal token ring networks. At time of system initialization, the first station on each ring receives the token. If there is any data to transmit, the station can send the data across the network to the destination station. Each station reads the message to determine if it is the receiver. It then passes the message onto the next station on the ring until it comes back to the source. If the data read is destined for the station, it passes the message to the node attached to that station. If there is no data to transmit, or the message transmitted comes back around to the source, the station releases the token and passes it to the next station.

When there is a message that has to go across a bridge to get to its destination on another ring, the network routes the message to the bridge on that ring. The bridge then holds the message until it receives the token on the other ring. When the token is received, the message is transmitted to the destination on that ring. This procedure is repeated if the message needs to go across multiple bridges to reach its destination.

2.2.1 Packet Layout

The packet layout used in this network is based on the IEEE 802.5 standard. However, in this network, some additional information is stored. As with the 802.5 standard, this layout contains the access control byte, source, and destination addresses. Along with their addresses, the source and destination's types are stored so that each node will know what type of node it is receiving data from. One piece of information that is stored in this packet which is not stored in the 802.5 standard, is a signal. This along with the source type will allow the destination to determine what kind of data it is receiving. Since each node sends different amounts of data to other nodes, each packet contains a field which stores the number of relevant bytes contained in the data field. This allows a node to only read what is necessary from the data field. The last field contained in this packet is the internal process

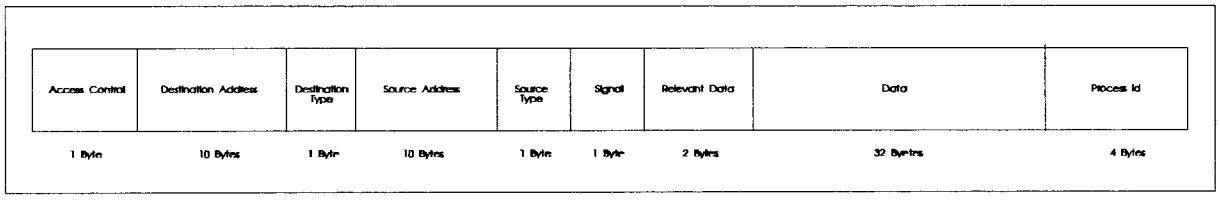

Figure 2-4: Network packet layout.

id to the simulation. This data is used by the monitor to track a process as it moves from resource to resource until it is complete. An example of this packet is displayed in Figure 2-4.

A brief description of each field is presented here:

- **Access Control.** The *access control* byte is used to identify whether the packet is a data packet or a token.
- **Addresses.** The *address* field is a 10 byte field used to determine the location of the source and destination of a station on the network. This address consists of a 5 byte ring number and a 5 byte ring address.
- **Types.** The *type* field is a single byte used to represent the type of node located at the corresponding address. This number corresponds to the list given in section 2.1.1.
- **Signal.** A *signal* is a field which when combined with the source type, allows the destination to determine what type of data it is receiving.
- **Relevant Data.** The *relevant data* field is a 2 byte integer field representing the number of bytes which are relevant to the destination.

- **Data.** The *data* field is the field which the source node fills with data in order to send it to the destination.
- **Process Id.** The *process id* field is a 4 byte field used to store the simulation's internal process id so that a process can be monitored as it moves from resource to resource.

2.3 Bridges

Bridges are those types of stations which are used to connect two rings to allow communications between stations on each ring. When a station wishes to send a message to a station on another ring, the network routes the message to the bridge that connects the two rings. The bridge then stores the message until it receives the token on the destination station's ring. Once it receives the token on that ring, it transmits the message to the station. If a message needs to go across multiple rings, the network uses the routing table to determine the path from one bridge to another until it reaches the destination.

In his book *Computer Networks*, Tanenbaum[2] describes two common implementation of bridges in the IEEE 802 standard. The two types of implementations are Transparent bridges and Source Routing bridges.

2.3.1 Transparent Bridges

The first 802 bridge is a **transparent bridge** or **spanning tree bridge**[3]. The basic theory behind this design is that a bridge can be added to a network and no hardware or software changes along with downloading of routing tables is required.

The transparent bridge operates in a promiscuous mode, accepting every frame transmitted on all the rings to which it is attached[2]. When a message arrives at a bridge, the bridge needs to decide whether to discard it, or forward it. This is done by looking up the destination address in a hash table in the bridge. This table lists each destination, and which ring it is on. From this, the bridge can decide if the message should or should not be forwarded.

When a new bridge is placed on the network for the first time, the hash tables are empty. At this time the bridge uses a flooding algorithm based on Baran's backward learning, to build its hash table. Every incoming message is broadcast on each ring to which the bridge is connected except for the one it arrives at. As time goes on, the bridge learns which destinations are connected to which rings and only broadcasts the message to the ring in which the destination is on.

2.3.2 Source Routing Bridges

Another type of bridge, preferred by the ring group of the IEEE 802 committee, along with the encouragement of IBM, is **source routing bridges.**

In this scheme the network assumes that the sender of each message knows whether or not the destination is on its own ring. When sending a message to a different ring, the station sets the high-order bit of the destination address to 1, to mark it. Also, it includes the exact path the message is to follow on the network. Therefore, a route is then a sequence of bridge, ring, bridge, ring, ... numbers. A source routing bridge is then only interested with the messages which have the high-order bit of the destination set to 1. For each one of these messages, it scans the route looking for the number of the ring on which the message arrived. If this ring number is followed by its own bridge number, the bridge forwards the message onto the ring whose number follows its bridge number in the route. If the incoming ring number is followed by the number of some other bridge, it does not forward the message.

2.3.3 Design Implementation

The design of bridges in this network are based on a combination of transparent bridges and source routing bridges.

Since this network is to be used with a predictable real-time system, the time it takes for a station to transmit a message over a bridge to a station on another ring must be predictable. By knowing the absolute path in which the message must travel, the upper bound arrival time of that message at it's destination can be calculated. This is obviously essential with the use of real-time systems. Based on this arrival time, the source station can determine if the arrival time will be less than the deadline time of the message. If it is not the message can be routed to a different resource in which the arrival time will meet the message deadline time.

In this simulation, when a station has data to send to a station on another ring, the network examines the message's destination address and determines, based on the routing table read in at simulation initialization, the bridge in which it needs to route the message to in order to reach the destination ring. That bridge holds the message until it receives the token on the next ring. When it receives the token, the network repeats the procedure of using the routing table to find the next bridge in which to route the message to. This is repeated until the message reaches the ring in which the destination station is located.

Since multiple messages can be sent to a bridge while its waiting for the token on another ring, each bridge maintains a queue of messages to be transmitted. This will prevent collisions of messages at a bridge.

An example of communications through bridges is presented here:

Recall Figure 2-2 in Section 2.1.1. If the station connected to node type 2 on ring 0 had data to transmit to node type 8 on ring 2, the following will occur:

1. When the station receives the token, the network analyzes the destination address.

2. The network determines that the message is destined for a station on ring 2 and looks up in the routing table which bridge on the source's ring the message should be routed to.

3. Since the routing table states that ring 0 must use bridge -1 to reach ring 2 it routes the message to that bridge.

4. When the bridge receives the token on ring number 1, it again analyzes the destination address and routes the message to bridge number -2.

5. Finally, when the bridge receives the token on ring number 2, it routes the message to the station connected to node type 8.

So basically, the design of bridges in this network uses the idea of transparent bridges in that the source does not have to know what path needs to be taken in order to reach the destination and, it uses the idea of source routing for the routing table so that the network can determine which bridge to use in order to reach the destination ring. However, instead of having the source station know the routing table, the bridges use the routing table to route messages throughout the network.

3 Network Operation

In this section, the actual methods of operation of the network are described. These methods are simulation initialization and simulation operation.

3.1 Simulation Initialization

At time of simulation initialization, the network will read in the topology and routing table, and build the network data structure. As each type is read in, a node instantiation of the type's object is created. This node is attached to the network's station. A pointer to the send and a pointer to the receive buffers are then passed to the object. This is the method in which the station and node communicate with each other. As the network reads the topology, it also creates a table of bridges and their ring addresses. This table is used for routing messages between rings until they reach their destination.

After reading in the network topology and building the bridges table, the network routing table is read in. As described in Section 2, this table is used to route messages through bridges over multiple rings.

Another table which is built at simulation initialization is the Network Geometry Table (NGT). This table is used by the monitor to get information from a station or a node located at a given address. When an instantiation is created a pointer to a reporting function for that node is passed to the network. The network takes the pointer to the node's reporting mechanism, the pointer to the station's reporting mechanism, the stations's address, and the node's type and creates the NGT.

Finally a token is placed at the first node on each ring and all variables are initialized in order for the simulation to start.

3.2 Simulation Operation

The operation of the network will require that the running simulation pass the network the simulation time for every clock tick of operation. At this time, the network will perform 3 major functions. These will consist of:

- Calculating delay times and message deadline time at a given station.
- Delivering any arriving messages to their destination.

- Checking for an arrival of a token at a station.

Also throughout the operation of the simulation, the network will provide any object a method of locating a resource on the network. This function will return the object with the address of the resource type it requests. This will be used mainly at the beginning of the simulation for objects to locate necessary resources on the network.

Prior to describing the operations of each of these 3 functions, the station to object communications will be defined.

3.2.1 Station-Object Communications

As described earlier in Section 3.1, when the network topology is read, each object is instantiated. As each object is instantiated, a pointer to it's send buffer and a pointer to it's receive buffer are passed to it by the network. These buffers can be written to and read by both the object and the network. An example of this is displayed in Figure 3-1.

So as not to overwrite data in a given buffer, the **access control field** is marked by

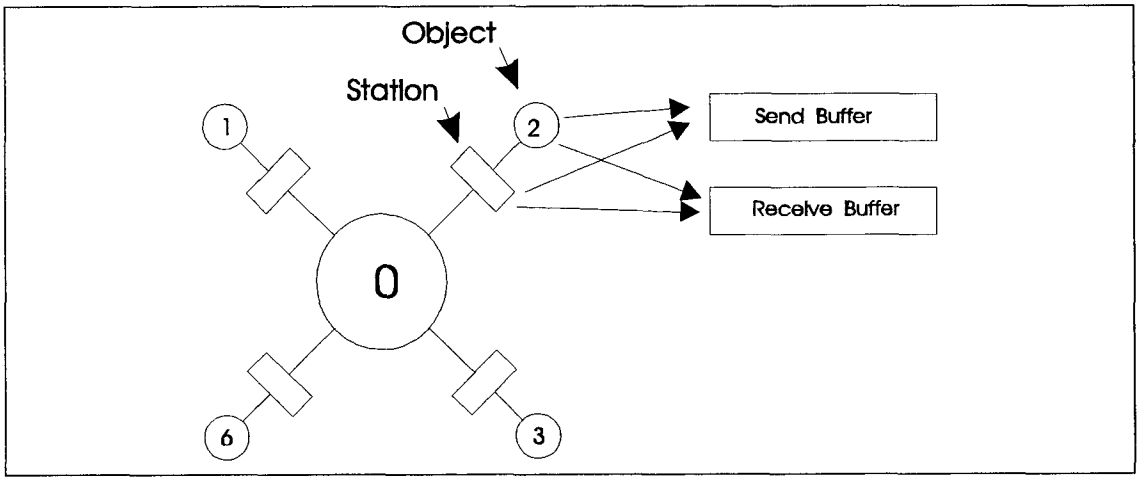

Figure 3-1: Station-Object communications.

either the object or the station to signal the other. The parameters for sending and receiving data are:

- SENT
- UNSENT
- READ
- UNREAD

For example, when an object wishes to send a message on the network, it checks it's send buffer to make sure that there are no messages waiting to be transmitted. It does this by checking the access control field in it's send buffer. If the field is marked as *SENT*, the object if free to fill in data and mark it as *UNSENT*. If the field is marked as *UNSENT*, this means there is a message waiting to be transmitted so the station needs to wait until the status changes. When the station receives the token, the network checks to see if that buffer has any data to be transmitted. It does this by checking if the access control field is marked *UNSENT*. If so, it copies the data and marks the field as *SENT* so that the object can transmit it's next message. If the field is marked as *SENT*, the object must not have any data to transmit, therefore, the station simply passes the token onto the next station.

Periodically the object will be required to check it's receive buffer for any incoming messages. It does this by checking the access control field in it's read buffer. If the field is marked as *READ*, then there is no incoming message. If the field however, is marked as *UNREAD* then the object can copy the data and mark the buffer as *READ*. In reference to the network, when an arrival occurs at a station, the network fills in the data in the receive buffer and marks it as *UNREAD*. Since this is a predictable real-time network, if there is a message which still has not been read by the object, the network will overwrite any data existing in the receive buffer.

3.2.2 Calculate Station Delay and Deadline Time

The first of the 3 major functions performed by the network will be calculating each station's **delay time** and each message's **deadline arrival time.**

For each clock tick of simulation time, each station on the network will be checked to determine if the station has any messages waiting to be transmitted. If an object has a message to be transmitted, it fills in the necessary fields in it's send buffer. It then marks the message as *UNSENT*. When the network is scanned for any waiting messages, it checks each station to see if it has any messages marked as *UNSENT*. If it finds one, the network assigns the current simulation time, which is the time the message was ready to be transmitted, to the start of delay time at the station. This is done for each station on the network. When the message is finally transmitted, the total delay time for that message at the station is then calculated and added to the total station delay time, total ring delay time, and total network delay time. These figures are then used to report to the monitor for network performance.

Since a requirement of this network to be used with a *real-time system* is **predictability**, the deadline message arrival time is calculated. This time represents the latest possible time the message can arrive at its destination. If the message cannot arrive by this time, it will no longer be relevant. When the station receives the token, the upper bound arrival time of the message at the destination is calculated. If the upper bound arrival time is greater than the deadline arrival time, the message must be routed to a resource at another location in which the arrival time at the destination will fall within the deadline arrival time.

3.2.3 Delivery Of Arriving Messages

The second function performed by the network for each simulation clock tick is the delivery of arriving messages at a destination. At each clock tick the network scans the data structure holding any messages on the network. If the arrival time of the message on the network is equal to the simulation time, then an arrival has occurred. The function takes the message off of the network, copies it to the destination station's receive buffer, and marks it as *UNREAD*. At this point the object can then read the data contained in the receive buffer and mark it as *READ* so that it can continue receiving messages. Since the **upper bound arrival** time is calculated when the token arrives at the station and the **deadline arrival** time is calculated when the message arrives at the station, it is known that the arrival time is less than or equal to the deadline time in that the message would have never been placed on the network if it would not arrive at the destination within the deadline time. In this case the message would have been routed to a resource located elsewhere on the network in order to arrive at the destination within the deadline arrival time.

Prior to actually delivering the message however, the network checks to see if the destination is a bridge on that ring. If the destination station is a bridge, the network will copy the message to the bridge's send buffer on the connecting ring.

Since a message can be transmitted to a bridge and the bridge then has to wait for the token on the connecting ring, before the network copies the message to the bridge's send buffer on the connecting ring, the network checks to see if there are any messages waiting to be transmitted on that ring. If there are, the network places the message in the bridge's waiting queue. When a bridge transmits the first message in it's waiting queue, the next message in the queue is then moved to the top.

On the surface, when using a queue at a bridge, one may suspect that the predictability of the network would not be maintained. However, since the ring latency and the queue length can be calculated it is possible to find the upper bound arrival time at a station. This calculation can be more accurately calculated every time a message is transmitted from the bridge.

For example, when a message arrives at a bridge to be transmitted on the connecting ring, an upper bound arrival time at its destination on that ring can be calculated. This is done by calculating the ring latency (assuming every station on the ring will have a message to transmit), and calculating the number of messages waiting before it in the waiting queue. However, when a message is transmitted from that bridge and the queue is updated, a new upper bound arrival time can be calculated for each message waiting in the queue. Since each station on the ring may not have a message to transmit when it receives the token, the ring latency will be reduced and thus the upper bound arrival time for each message will be also reduced. This will assist in maintaining acceptable network performance while maintaining arrival time **predictability**.

3.2.4 Arrival Of Token At A Station

The third function performed by the network is checking for an arrival of the token at a given station on each ring.

When the simulation starts, the arrival time of the token at the first station on each ring is set to 0. Then, when this function is called, it checks to see if the token has arrived at any station on the ring. From this it is observed that the first station will have the token.

The network then checks to see if the station which has possession of the token, has any data in its send buffer marked as *UNSENT*. If there is data there waiting to be transmitted, the network checks the deadline arrival time and calculates the arrival time of the message at the destination. If the arrival time is less than or equal to the deadline arrival time, the network copies the contents of the send buffer, places it on the network, and marks it as being *SENT* at the station's send buffer. If the **deadline arrival** time cannot be met, a path to an alternate resource of the same type, in which the arrival time will meet the deadline arrival time, is found and the message is routed to that location. This will insure that the constraint of deadline arrival time for real-time systems and **predictability** of message arrival is met.

Prior to placing the message on the network, the function determines if the destination of the message is on the current ring, or if the destination is located on another ring attached to the network. If the destination is on the current ring, the network simply calculates the arrival time of the message and places it on the network.

If the destination is not on the current ring, the network has to determine which bridge on the current ring to send the message to so that it can reach it's destination. At this point it references the network routing table. Since it knows which ring the destination is on, it looks up that ring in its routing list. Attached to that ring is the local bridge which the message must be routed to. When the network receives this bridge number, it locates its address in the bridge list and routes the message to that bridge.

By using the routing table, a path through bridges to the destination and an upper bound arrival time is calculated. With this upper bound arrival time, the network can insure that a message will arrive at the destination station within given time restraints. This enables the network to display the qualities of predictability required by real-time systems.

Upon placing the message on the network, the return time of the token at the originating station is calculated and then the arrival of the token at the next station is calculated and stored in the ring information table.

If the station has no data to transmit, the network simply calculates the arrival time of the token at the next station on the ring and passes the token on.

3.2.5 Location Of Object Type

Another function provided by the network is a method of locating a given resource on the network. This function can be called by all the objects on the network. Given an object type described in section **2.1.1**, the network will return the address of that object on the network if it exists. This will provide a means for starting the simulation. For example, if a CU needs to have something performed by an ALU but does not know where one is on the network, it can simply call this function and it will return the address of an ALU on the network.

This network resource is also used for finding alternate resources on the network when a message arrival time exceeds the constraints provided by the deadline arrival time. If a station sending a message to another, determines that the calculated arrival time exceeds the deadline arrival time, the station can use this

resource in order to find a station of the same type in which the arrival time will be within the constraints of the deadline time.

This can also be used by the schedulability analyzer. A *schedulability analyzer* is used to analyze a program for predictable schedulability. That is, every program expressed in the language may be analyzed statically for its adherence to the timing constraints. It consists of two parts: a partially language-dependent front end, and a language-independent back end[31]. The front end is used to extract the code being generated, timing information, and calling information. The back end is used to correlate all information gathered in recorded in program trees by the frond end, and predict guaranteed response times for the entire real-time application[31].

Upon creating a schedulability analyzer, this function can be used by the analyzer to schedule operations which need to be performed at a given object. The analyzer can use this function to determine where each type of object is located on the network. Then, based on how busy a given resource is, it can schedule operations at other resources on the network until that one is available.

4 Reporting Mechanisms

This section will describe the performance information for the network which is reported to the monitor. The three main aspects of the network which can be studied include the entire **network**, a given **ring** on the network, and a given **station** attached to a ring on the network. From the information obtained from the reporting mechanism, an optimal network topology can be determined based on transmission delay times, average bridge queue size, and total messages received and sent. From this guidelines for creating a network for a distributed real-time system can be obtained. Also, properties of high throughput and predictability are maintained.

4.1 Station Reporting

Calculating network performance as it relates to a station on the network will assist in determining the optimal network topology. For example, if resources which are more frequently used are all placed on one ring, average delay times of messages waiting to be transmitted can increase significantly. Therefore, a facility is needed to determine how frequently a given resource is used.

The pertinent information which will be described by this reporting mechanism will consist of:

- Total messages sent by station
- Total messages received by station
- Average and worst-case message delay time
- Total messages exceeding deadline

4.1.1 Total Messages Sent By Station

In order to calculate the delay time of a message at a station, the number of messages transmitted by that station needs to be recorded. However, this is not the only purpose which this calculation serves. In order to determine the optimal placement of resources on a network, the frequency with which these resources send data needs to be determined. If resources which are frequently sending data are on the same ring, delay times can increase significantly because each station on that ring will have to wait until every other station on that ring has transmitted its message before it receives the token again. With real-time systems this

significant delay times are not tolerable. Therefore, by calculating the total number of messages sent by a station, guidelines to placing resources on the network can be created so as to minimize delay times on a given ring.

4.1.2 Total Messages Received By Station

By calculating the total number of messages a station on the network receives, the guidelines for resource placement discussed in the previous section can be further enhanced.

If for example, a station which has a high frequency of messages sent is placed on one ring, and a station which has a high frequency of messages received is placed on another, it can be determined that the bridge connecting those two rings will maintain large queues of waiting messages. This will not affect waiting time of a message at a station however, this will affect arrival time of a packet at its destination.

Since these two stations are constantly sending messages between each other, the bridge connecting these two rings will be flooded with messages to transmit on each ring. Therefore, by calculating the number of messages a station receives, we can determine which resources should be grouped together on the same rings. This will reduce the amount of use of a bridge therefore maintaining a consistently quick arrival time of a message at its destination.

4.1.3 Average and Worst-Case Message Delay Time

The delay time of a message at a station is given by calculating the difference of when the message arrived at the station to be transmitted and when the message was actually transmitted. When analyzing the network performance, the average delay time a message at a station experiences is calculated in order to determine and then optimize the amount of stations placed on one ring.

The nature of a real-time system requires that the average delay time be as low as possible. Therefore, by calculating the average delay time a station experiences before receiving the token, it can be determined how many stations should be placed on one ring.

However, with ring networks there is a **worst-case** delay time in which a station will receive a token. This delay time can be determined by calculating the ring latency assuming every station has a message to transmit. If every station on the ring has a message to transmit, then the worst-case time before a station receives

the token again is simply calculated by multiplying the ring latency by the number of stations on that ring.

By running a number of simulations with different topologies, the average delay time a station experiences and the average delay time a bridge experiences will determine if the network should be laid out with more rings and less stations per ring, or minimizing the number of rings while maximizing the number of stations per ring.

4.1.4 Total Messages Exceeding Deadline

In order to determine what effect station delays have on a real-time system, the number of outgoing messages which will not arrive at the destination within the deadline arrival time is recorded. If a station experiences unusually high delay times, the efficiency of the network will be greatly reduced. While although maintaining predictability, this performance will not be suitable for a real-time system. By studying what effects different network topologies have on this variable, it will be possible to determine a worst-case network topology and it will help to obtain an optimal one.

4.2 Ring Reporting

As it relates to a ring, network performance can determine guidelines for ring use. Since each ring on a network maintains its own token, it is possible for some rings to experience more traffic than others. In the case of real-time systems, we would like to distribute the resources on the network in such a way as to balance the traffic on each ring. Therefore, in order to study this, three performance measures will be calculated for each ring on the network. These three measures consist of:

- Number of messages transmitted on a ring
- Number of messages received on a ring
- Average and worst-case delay time of stations on a ring

4.2.1 Messages Transmitted and Received

In determining the number of messages transmitted and the number of messages received on a ring, the amount of traffic a ring experiences can be calculated. For example, if the number of messages transmitted on a ring is significantly larger than the number of messages received on the ring, it is obvious that a large percentage of the messages transmitted are crossing over to another ring via a bridge. As described in section **4.1.2**, if a bridge experiences a large amount of

messages needing to cross over to another ring, average network delay time increases and packet arrival time increases. This can prove to be unacceptable in a real-time system. So, with the use of number of messages transmitted and number of messages read, an optimal ring layout can be determined.

4.2.2 Average and Worst-Case Delay Time of a Ring

In order to reduce the amount of time a station has to wait for a token on a ring, it is generally determined that the number of stations on that ring should be reduced. In the case of a real-time system, it is required that a maximum average and worst-case delay time be calculated. From this, the network topology can be designed in order to meet these requirements. In some cases it may be preferable that the average delay time be as low as possible. In this case, the number of stations on a ring should be minimized. However, if the average delay time is a medium range number, then number of stations on a ring can be increased. Also, a given real-time system may set a worst-case delay time. By calculating the average and worst case delay times experienced on a ring, a designer can determine whether the number of stations on that ring should be reduced or increased. For the sake of this simulation, a number of simulations can be run to determine how the number of stations on a ring affects the average delay time. This along with the worst-case delay time, will in essence, assist in determining if the network should be laid out with a minimal number of stations per ring or if there is a limit as to how many stations can be placed on a ring before affecting the performance of the network.

4.3 Network Reporting

When designing this network, certain performance requirements were taken into consideration.

1. The network has to be predictable.

In order for a network to be predictable, a message must arrive at its destination when it is supposed to. Also, a station must be able to determine at what time it will receive the token in order to transmit a message.

2. The network must provide reliable performance.

Since this network is designed for use with distributed real-time systems, it needs to provide performance which is reliable and efficient.

Therefore, taking these two requirements into consideration, average network delay must be reduced and throughput must be maximized. If these two goals can be met, then this network will provide a suitable means of communications for the resources which make up this real-time system.

4.3.1 Predictability

As described in Section 3.2, this network provides **predictable** performance. First, for a given set of performance parameters, it can be proven that this network is predictable in that the time it takes for any operation to be performed by this network can be calculated. Even before the network is placed into operation, the three measures which prove the predictability of this network are the ability to **(1)** calculate the arrival time of a token at a station; **(2)** calculate the arrival time of a message at a destination and; **(3)** determine whether a message will arrive at a destination within the deadline arrival time constraint.

As described in Section 3.2.4, based on the network topology and token position, the upper bound arrival time for a token at a station can be calculated. This is done by calculated the ring latency, assuming each station has a message to transmit, and multiplying this by the number of stations on the ring.

Since the network can determine the absolute path taken by a message in reaching its destination, the arrival time of the message at a station can be calculated. Due to this, a station can determine if an outgoing message will arrive at its destination before the deadline arrival time. If the message will not, the station can send the message to an alternate destination of the same type.

For this reason, the average and worst-case delay time and throughput of this network will be studied. By running simulations using different network topologies, determinations of how placement of stations and number of rings affect the performance of the network. From these studies, rules on creating network topologies can be determined.

5 Module Design

This section will describe in detail the functions performed by the modules which make up this network.

5.1 Read Topology

Recall Figure 2-1. The read topology module reads the network topology file defined in this Figure. From this file it builds the network topology data structure and assigns each station its network address. As it reads in the topology file, it determines if the number read in corresponds to a bridge. If so, it creates an entry in the bridge table and enters the current ring address. When the corresponding bridge is read in it adds the bridge's address on the connecting ring as the second address in the bridge table. Upon reading in a station, the module creates an instantiation of the object corresponding to the number read in. Upon creating the instantiation, the network passes pointers to the send and receive buffers in order to establish communications between the object and the station. When the instantiation is created, the object receives the two pointers and passes a pointer to the reporting function for that object. The module then receives this function and along with the station's reporting function, station's address, and object type,

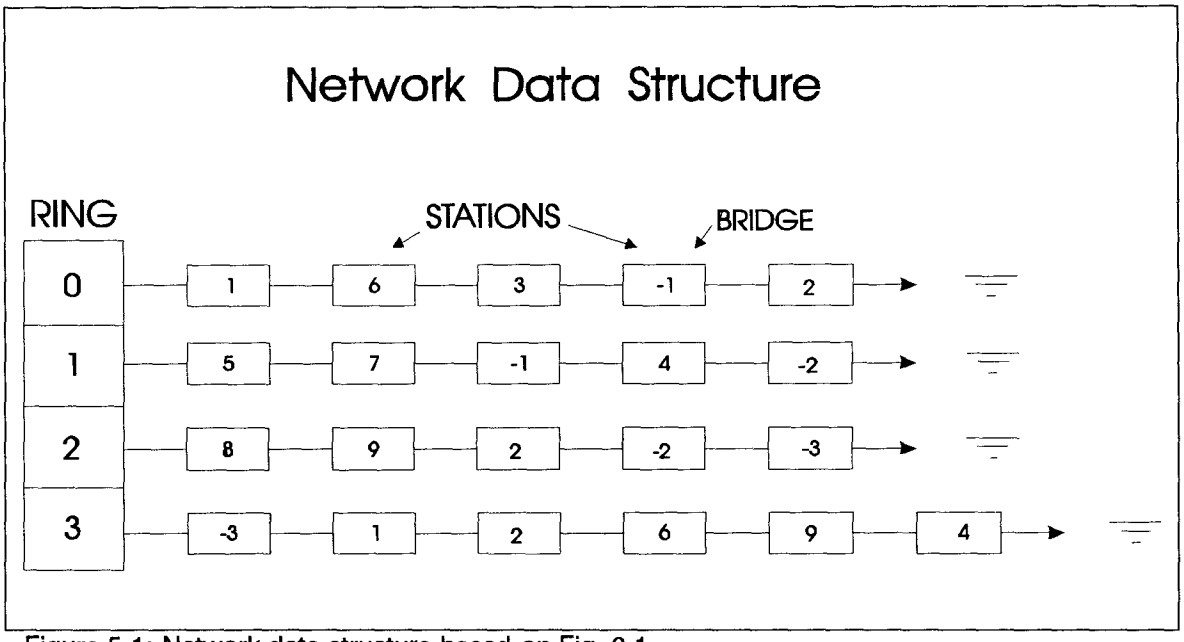

Figure 5-1: Network data structure based on Fig. 2-1

creates and entry in the Network Geometry Table(**NGT**) defined in section 3.1.

A sample of the data structure corresponding to the topology file in section 2.1 is displayed in Figure 5-1.

After reading in the network topology file, the module initializes all variables and places the token at the first station on each ring in order to start the simulation.

5.2 Read Routing Table

Recall Figure 2-3. The read routing table module reads in each entry to the routing table defined in section 2.1.2, and creates the routing table data structure used by the network to route messages throughout the network to their respective destinations. Each entry consists of the destination ring and the local bridge used to reach that ring. The module starts with ring 0 and reads in each table entry. A -99 is used as the ring delimiter to increment the ring number by 1 until the file is completely read.

5.3 Update Network

The update network module is the module which performs the network operations. This module is called for every clock tick of simulation time and is passed the current simulation time.

When the function is called, it first checks for the arrival of a message to be transmitted at a station. It scans the network data structure defined in Figure 5-1. If a station has a message to be sent, it assigns the simulation time to the message delay start time. This is used later to calculated the average message delay time at a station.

After scanning the network data structure for any messages waiting to be transmitted, the module scans the data structure containing all messages currently traversing the network for any arrivals at a station. A sample of this structure is displayed in Figure 5-2.

If a message on the network arrives at its destination, the module copies the message to the station's receive buffer and removes it from the network. If the message arrives at a bridge, the module places the message in the bridge's send buffer on the connecting ring so that when it receives the token on that ring, the

message can be transmitted. It then calculates the return time of the token at the source and the arrival time of the token at the next station on the network.

Finally the module checks for an arrival of a token at a station. If the station has data to transmit, the module calculates the arrival time of the message at its destination and copies the packet on to the network data structure displayed in

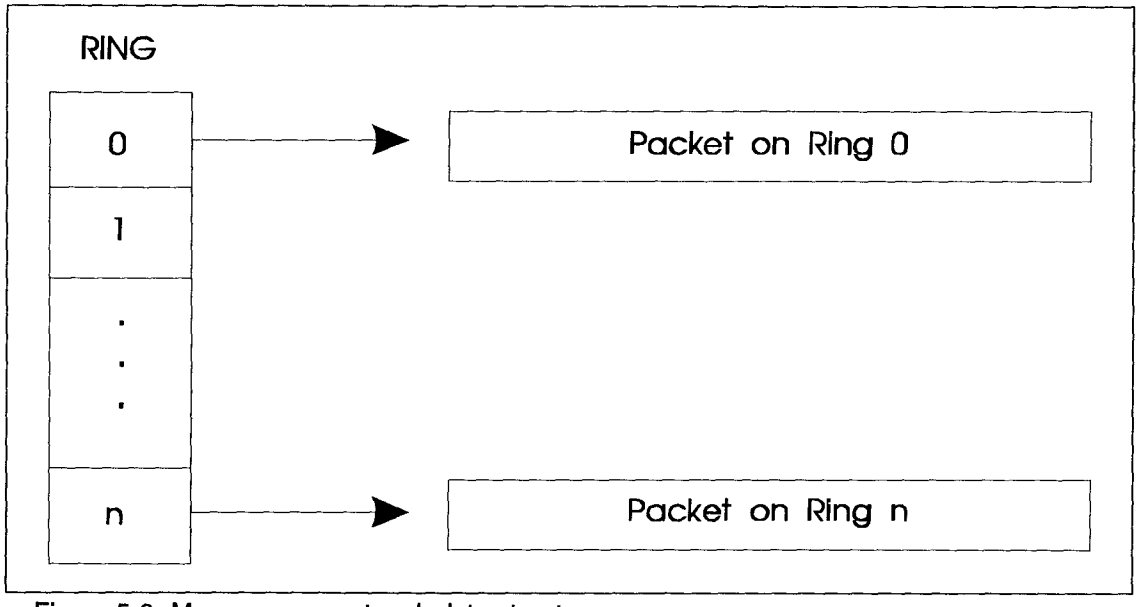

Figure 5-2· Messages on network data structure.

Figure 5-2. If the station does not have any message to transmit, the module calculates the arrival time of a the token at the next station on the ring and simply passes the token on.

5.4 Send Monitor

The send monitor module is used to send the network topology to the monitor when requested. When a monitor requests the network topology, this module reads the network data structure and passes the addresses and types one by one to the monitor. Each address is also passed with a code corresponding to the type of entry it is. If the address is the first in the network topology, the address and type are accompanied by an *NTSTART* flag to signify to the monitor that this is the first address in the network topology. A normal entry is accompanied by an *NTENTRY* and the last address and type are accompanied by a *NTEND* to instruct the monitor that this is the last address in the network topology. From these

addresses and types the monitor can build the network topology to be displayed on the screen.

5.5 Station Report

The station report module is used to report statistics concerning the selected station. This module is called from the a monitor based on the **NGT**. The reported data are the station address, station type, and station latency. Also, this module calculates the number of messages transmitted and received by this station. Using these calculated numbers, the module calculates the average station delay and calculates the total number of messages which exceeded the deadline arrival time. With the monitor, a given station's average delay time and utilization can be observed.

Also, this module can report on queue lengths at a bridge or how many messages have traversed over a given bridge throughout the simulation.

5.6 Ring Report

The ring report module which is called directly from a monitor, is used to report all statistics concerning a given ring's operation. This module calculates the average ring delay and the ring latency and reports the information to the monitor. From these figures, the traffic on the ring can be monitored to determine how the reduction or increase of the number of stations on the ring would effect ring performance.

Along with passing ring traffic information, this modules reports token position and message position on the ring. With this, someone using the monitor can track a token or message as it traverses the ring.

5.7 Network Report

This module reports network performance measures to the monitor. As with the ring reporting module, this module is directly called by the monitor. When called, it calculates network utilization and average network delay and passes the information to the calling monitor. From this a user can view the number of messages meeting and not meeting deadline arrivals in a real-time environment.

6 Evaluation

In order to evaluate the effectiveness in using this network for a real-time system over a conventional network, this network needs to prove two requirements in order for it to be considered better than a conventional network. These two requirements are:

1. The network needs to be predictable and,

2. The network needs to reduce delay times experienced in conventional networks.

This section will prove that the network defined in this paper is predictable and more efficient than conventional networks. Through the use of extensive simulations performed, this section will show why this network is suitable for use with predictable real-time systems.

6.1 Predictability

As it has been proven in Section 3 and Section 4 of this paper, this network is predictable in all phases of operation. To prove this, it has been shown that 3 measures needed to make this predictable could be calculated during execution. These three measures are:

1. Arrival time of a token at a station.

2. Arrival time of message at a destination.

3. Determination of message arrival before deadline arrival time.

Since the network topology is known, by calculating the ring latency and knowing the position of the token on a ring, the upper bound arrival time of a token at a given station on a ring can be calculated. For every movement of the token through the ring, this number can also be re-calculated to find the actual arrival time of the token at the station.

When sending a message on the network, the arrival time of the message at the destination can also be predicted. Since the network topology is know, when a

message is to be transmitted to a destination, the routing table along with the network topology and the bridge table can be used to calculate the upper bound arrival time of a message at its destination. With this one can determine the worst-case arrival time at the destination. This can then be compared to the deadline arrival time to insure that the message will arrive at the destination in time.

By determining the deadline arrival time, and calculating the worst-case arrival time of a message at its destination, the network can then determine whether the message should be transmitted to the destination or not. This will rid the network of wasted messages, decrease average waiting time for a token, and improve network performance.

6.2 Efficiency For Real-Time Systems

Although a network may be predictable, it may not be well suited for use with a real-time systems. If arrival times of messages go beyond that of the deadline time, the message becomes useless and just inhibits performance of the network. In order to determine that this network is well suited for predictable real-time systems, it must be proved that **(1)** a station will not experience any unusually long delay times in waiting for a token and that **(2)** messages will arrive at their destination before their deadline time. Although it is impossible to insure that every message will arrive at its destination before its deadline time, it is preferable that the number of messages which do not arrive at their destination before their deadline time, be minimized.

Several advantages which this network has over conventional networks which make it more suitable for real-time systems are:

1. This network uses a multiple token ring architecture which allows the number of stations to be divided over different rings. This increases the frequency with which a token arrives at a station. This in turn, reduces delay times at a station which increases network performance.

2. By decreasing the delay time experienced at a station, the likelihood of a message arriving at its destination within the deadline time is increased.

3. Also, by using a multiple token ring network, a more accurate upper bound arrival time can be calculated reducing the number of messages which will not arrive by their deadline time.

4. Since most conventional networks are not predictable, messages which will not reach their destination within the deadline time, are transmitted anyway increasing network traffic and station delay. With this network, since the arrival time can be predicted, if the arrival time of a message is greater than the deadline time, the message can either be discarded or routed to another station in which the arrival time falls within the deadline time.

6.3 Performance Comparison

In order to prove the efficiency of this network, a performance comparison has been done versus a conventional token ring network.

The evaluations for each of the comparisons were done by running a number of simulations which created random arrivals and destinations of messages at a station. This arrival rate was varied from .2 messages arriving per quantum to 1 message arriving per quantum. The average station delay and number of messages exceeding their deadline was totaled and plotted on a graph to view the differences

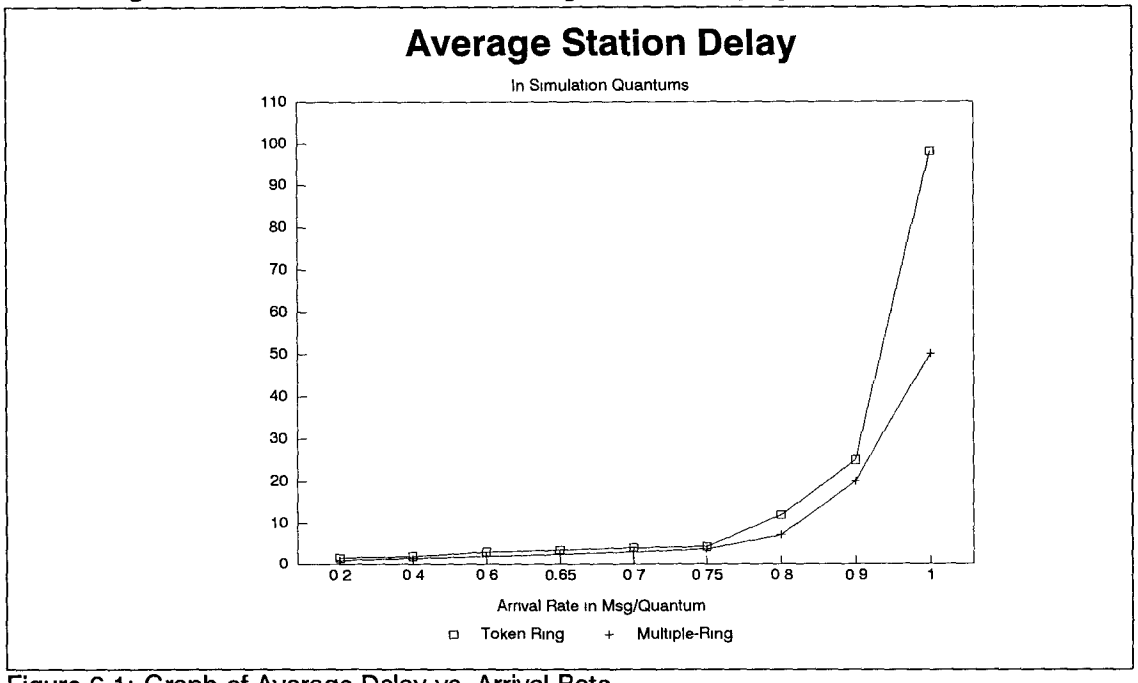

Figure 6-1: Graph of Average Delay vs. Arrival Rate.

experienced by the two networks. The findings and results are presented here.

The first comparison of the two architectures studies the average delay experienced by the two networks under the same traffic load. It has been found that as rate of arrivals increases at a station, the conventional token ring network experiences significantly higher delays at each station. A graph of arrival rates versus delay times is displayed in Figure 6-1.

In this simulation, the number of stations, station latency and station to station propagation time were held constant. As it can be observed, for low arrival rates, the conventional token ring experiences slightly higher delay times. However at an arrival rate of 1 message per quantum of time, the delay experienced by the multiple token ring network presented in this paper, is nearly half of the delay of the conventional token ring network. Since the conventional token ring network maintains only one token, as the arrival rate increases, most stations will have a message to transmit. Due to this, delay times will increase because the token is constantly propagating completely around the network. With the multiple token ring network, the stations are distributed over multiple rings and a token is maintained for each ring. With the multiple tokens, the rings are then smaller and

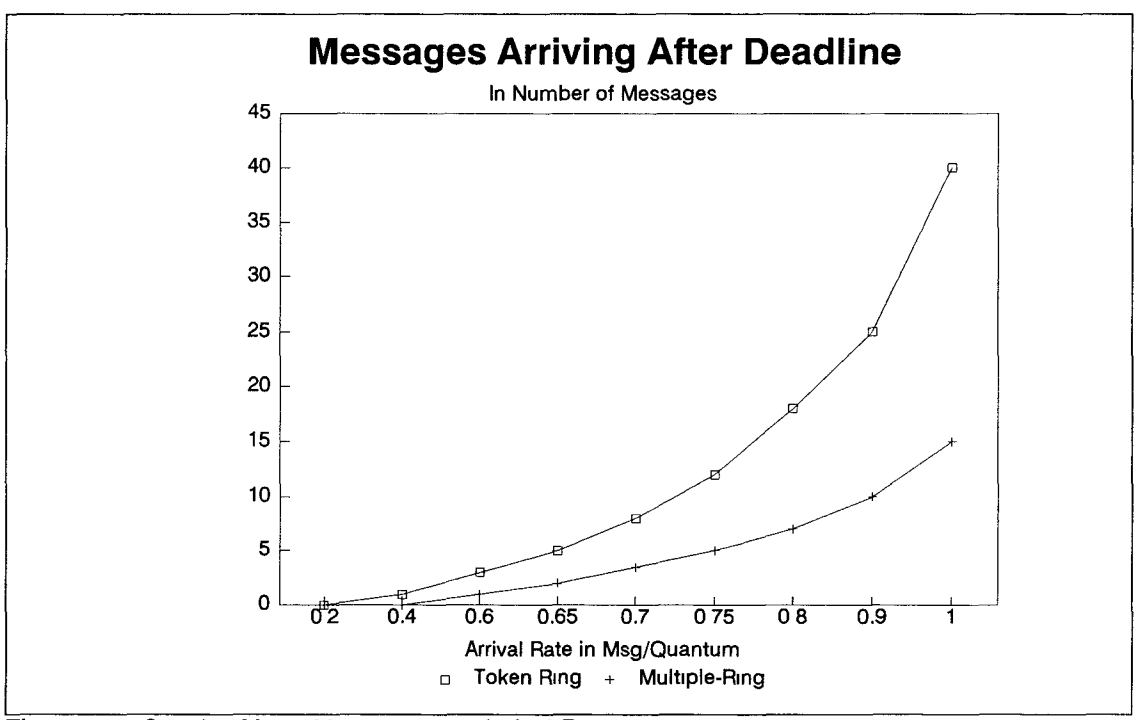

Figure 6-2: Graph of Lost Messages vs. Arrival Rate.

the frequency with which a station receives a token is increased.

In studying the worst-case possibility, it is observed that a station on a conventional token ring network would have to wait twice as long as a station on a multiple token ring network. Since the network is to be used with a real-time system, it is obvious that the conventional token ring network will not provide the performance necessary to deem it suitable for this type of implementation.

In the next simulation, the number of messages which do not arrive at their destination before their deadline is studied. From this study, it has been observed that as the arrival rate of messages at a station increases, the number of messages which are transmitted and do not arrive at their destination within the deadline time increases significantly for conventional token ring networks. A graph displaying this observation is displayed in Figure 6-2.

In this simulation, as in the previous, the number of stations, the station latency and the station to station propagation time were held constant. As it is observed, at high arrival rates, the conventional token ring network experiences more lost messages due to deadlines than the multiple token ring network presented in this paper. Although an arrival rate of 1 message may be an extreme case, it is beneficial to note that the multiple token ring provides reliable efficient performance.

In analyzing a worst-case situation in this simulation, it has been found that a conventional token ring network loses nearly twice as many messages due to deadline arrival time constraints. Since the traffic in a multiple token ring network is distributed over a number of rings, it is demonstrated that the number of messages lost due to deadline arrival times is reduced. This again shows the advantage the multiple token ring network has over the conventional token ring network.

6.4 Conclusions

As it has been demonstrated in this section, this network provides predictable and efficient communications for real-time systems. It enjoys several advantages over conventional networks and provides a basis for developing communications networks for distributed real-time systems.

7 Future Work

One aspect of network communications which was not included in this design is the idea of high and low priority messages. In this design all messages have equal priority when transmitting over the network. However, in some situations, the ability to obtain a token and send a high priority message, may be needed. Some network architectures, such as the Token Ring Scheduling Protocol described in chapter 1, provide a means of sending a high priority message.

With some modifications to the protocol used in this design, the ability to send high priority messages can also be implemented. Also, the implementation of this architecture lends itself to change easily. By simply creating a new protocol, this architecture can be modified to divide the bus into multiple channels, or a different protocol, such as CSMA/CD, can be implemented to see how it performs in a real time environment and to analyze its predictability.

REFERENCES

[1] Y.H. Lee, L.T. Shen, "Real-Time Communication in Multiple Token Ring Networks," *Proceedings of the IEEE 1990 Real-Time Systems Symposium* , 1990, pp. 146-153.

[2] A.S. Tanenbaum, *Computer Networks*, 2nd. Edition, Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[3] F. Backes, "Transparent Bridges for Interconnection of IEEE 802 LANs," *IEEE Network Magazine*, vol.2, pp. 5-9, Jan. 1988.

[4] W. Stallings, *Data and Computer Communications*, 2nd. Edition, Macmillan Publishing Company, New York, New York, 1985.

[5] M. Schwartz, *Telecommunication Networks*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1987.

[6] K.G. Shin, C.J. Hou, "Analysis of Three Contention Protocols in Distributed Real-Time Systems," *Proceedings of the IEEE 1990 Real-Time Systems Symposium*, 1990, pp. 136-145.

[7] J.K. Strosnider, T.E. Marchok, "Responsive, deterministic ieee 802.5 token ring scheduling," *Real-Time Systems Journal*, September 1989.

[8] C.H. Chen, L.N. Bhuyan, "Design and Analysis of Multiple Token Ring Networks," *INFOCOM 88*, 1988, pp. 477-486.

[9] J.A. Stankovic, "Real-Time Computing Systems: The Next Generation," *Real-Time Systems*, Feb 18, 1988, pp. 14-36.

[10] Sauer, Macnair, *Simulation of Computer Communication Systems*, Prentice Hall, New York, 1983.

[11] J.T. Martin, *Design of Real-Time Computer Systems*, Prentice Hall Series in Automation Computing, Prentice Hall, Englewood Cliffs, New Jersey, 1967.

[12] J. Magee, J. Kramer, "Dynamic Configuration for Distributed Real-Time Systems," *Proceedings of the IEEE 1983 Real-Time Systems Symposium*, 1983, pp. 277-288.

[13] A.S. Sethi, T. Saydam, "Performance Analysis of Token Ring Local Area Networks," *Computer Networks and ISDN Systems*, Vol.9, No.3, March 1985, pp. 191-200.

[14] J.K. Strosnider, T. Marchok, J. Lehoczky, "Advanced Real Time Scheduling Using the IEEE 802.5 Token Ring," *Proceedings of the IEEE 1988 Real-Time Systems Symposium*, 1988, pp. 42-45.

[15] J.F. Hayes, *Modeling and Analysis of Computer Communication Networks*, Plenum Press, 1984.

[16] P.C. Lee, S.P. Schaff, T.Y. Tsai, N. Srinivasan, "Design and Development of a Real-Time System - A Case Study", *Procedings of the IEEE 1982 Real-Time Systems Symposium*, 1982, pp. 184-194.

[17] W.A. Halang, A.D. Stoyenko, *Constructing Predictable Real-Time Systems*, Kluwer Academic Publishers, Dordrecht-Hingham, 1991.

[18] R.M. Cohen, "Formal Specifications for Real-Time Systems", *Texas Conference on Computer Systems, IEEE*, 1978, pp. 1-8.

[19] P.G. Sorenson, V.C. Hamacher, "A Real-Time System Design Methodology", *INFOR*, Vol. 13, No. 1, February 1975, pp. 1-18.

[20] W. Bux, "Local Area Subnetworks: A Performance Comparison," *IEEE Transactions on Communications*, Vol. COM-29, No.10, Oct. 1981, pp. 1465-1473.

[21] I. Chlamtac, A. Ganz, "Design and Analysis of Very High-Speed Network Architectures," *IEEE Transactions on Communications*, Vol. 36, No. 3, March 1988, pp. 252-262.

[22] J.F. Kurose, M. Schwartz, Y. Yemini, "Multiple-Access Protocols and Time-Constrained Communication," *ACM Computing Surveys*, Vol. 16, No. 1, pp. 43-70.

[23] P. Baran, "On Distributed Communication Networks," *IEEE Transactions on Communication Systems*, Vol. CS-12, pp. 1-9, March 1964.

[24] D. Bertsekas, R. Gallager, *Data Networks*, Prentice Hall, Englewood Cliffs, New Jersey, 1987.

[25] T.Y. Choi, "Formal Techniques for the Specification, Verification, and Construction of Communication Protocols," *IEEE Communications Magazine*, Vol. 23, pp. 46-52, 1985.

[26] R.C. Dixon, D.A. Pitt, "Addressing, Bridging, and Source Routing," *IEEE Network Magazine*, Vol. 2, January 1988, pp. 25-32.

[27] M.C. Hamner, G.R. Samsen, "Source Routing Bridge Implementation," *IEEE Network Magazine*, Vol. 2, January 1988, pp. 10-15.

[28] J.Y-T. Leung, T.W. Tam, and G.H. Young, "On-Line Routing of Real-Time Messages," *Proceedings of the IEEE 1990 Real-Time Systems Symposium*, 1990, pp. 126-135.

[29] D.A. Pitt, "Standards for the Token Ring," *IEEE Network Magazine*, vol. 1, pp. 19-22, January, 1987.

[30] W.M. Seifert, "Bridges and Routers," *IEEE Network Magazine*, vol. 2, pp. 57-64, January, 1988.

[31] A.D. Stoyenko, V.C. Hamacher, and R.C. Holt, "Analyzing Hard-Real-Time Programs For Guaranteed Schedulability," *IEEE Transactions on Software Engineering*, vol. 17, no. 8, August, 1991, pp. 737-750.

Appendix

```
# -g include debug information

OBJ = tokenring.o readtop.o sendmonitor.o locatenode.o routingtable.o findbridge.o updatenet.o ringreport.

tokenring: $(OBJ)
        CC $(OBJ) -o tokenring -lX11 -lm

tokenring.o:    tokenring.c tokenring.h
        CC -c tokenring.c

readtop.o:      readtop.c tokenring.h
        CC -c readtop.c

sendmonitor.o:  sendmonitor.c tokenring.h rptmech.h
        CC -c sendmonitor.c

locatenode.o:   locatenode.c tokenring.h
        CC -c locatenode.c

routingtable.o: routingtable.c tokenring.h
        CC -c routingtable.c

findbridge.o:   findbridge.c tokenring.h
        CC -c findbridge.c

updatenet.o:    updatenet.c tokenring.h
        CC -c updatenet.c

ringreport.o:   ringreport.c tokenring.h
        CC -c ringreport.c

stationreport.o: stationreport.c tokenring.h
        CC -c stationreport.c

networkreport.o: networkreport.c tokenring.h
        CC -c networkreport.c

copypacket.o: copypacket.c tokenring.h
        CC -c copypacket.c

testnetwork.o: testnetwork.c tokenring.h
        CC -c testnetwork.c

depositvals.o: depositvals.c tokenring.h
        CC -c depositvals.c
```

```
clean:
	rm $(OBJ)
```

```
/*************************************************************/
/* Charles Silva                                             */
/* Master's Thesis                                           */
/* Communications Network for Distributed Real-Time Systems  */
/*************************************************************/
/* Main program to call all sub-programs.                    */
/*************************************************************/

#include "tokenring.h"

int     bridgeCount;
int     numberOfRings;


void    StuffTopology(char *, int *, int *);
void    MonitorTopology(int *);
void    LocateNode (int *, int *, int *);
void    ReadRoutingTable(char *);
void    TestNetwork();

void
main( int argc, char *argv[])
{
        struct stationStructure         *current;
        int index=0,
                NumberOfNodes=0,                /* Total number of elements read in including delimiter */
                NumberOfRings=0 ;       /* Number of Token Rings on the Net */
        int     lType=0;
        int     lRing=0;
        struct routingEntry *currentRoute;
        int     lAddress=0;
        int     currentring=0;


        StuffTopology( argv[ 1 ], &NumberOfNodes, &NumberOfRings ) ;
        ReadRoutingTable( argv[ 2 ] );
        printf( "Elements = %d   Rings = %d \n", NumberOfNodes, NumberOfRings ) ;


        MonitorTopology( &NumberOfRings );

        while (currentring <= numberOfRings)
        {
                currentRoute = routingTable[ currentring ].routeEntry;
                while ( currentRoute != NULL )
```

```
            {
                    printf("Ring Number-> %d  Destination Ring-> %d   Bridge Number-> %d\n",currentrin
                    currentRoute = currentRoute->nextRoute;
            }
            currentring++;
    }


    lType = 5;
    LocateNode ( &lType, &lRing, &lAddress );

    printf( "Locate Type = %d     Address = %d %d\n",lType,lRing,lAddress );


    TestNetwork();


}
```

```
/*************************************************************/
/* Charles Silva                                             */
/* Master's Thesis                                           */
/* Communications Network for Distributed Real-Time Systems */
/*************************************************************/
/* Header file of all data structures and variables used by */
/* functions related to the network.                         */
/*************************************************************/

#include <stdlib.h>
#include <malloc.h>
#include <string.h>
#include <stdio.h>
#include <ctype.h>

#define TRUE            1
#define FALS            0
#define MAXNODES        200
#define DATALENGTH      5000
#define NODEID          3       /* Size of the filed Includes NULL char */
#define NODERING        6
#define NODEADDRESS     6
#define ENDOFRING       0
#define MAXRINGS        100
#define MAXBRIDGES      100
#define CONTROLSIZE     1
#define UNREAD          0
#define READ            1
#define UNSENT          0
#define SENT            1
#define PACKETLENGTH    32

#define CU              1
#define ALU             2
#define REGISTERS       3
#define PSW             4
#define IOP             5
#define MAINMEM         6
#define SECMEM          7
#define COMMPROC        8
#define OSKERNAL        9
#define ENDROUTE        -99

extern int      bridgeCount;            /* Number of bridges on network */
extern int      numberOfRings;          /* Number of rings on network */
```

```
struct topologyData
{
        int nodeId;
        int nodeRing;
        int nodeAddress;
 } Topology[ MAXNODES ];


typedef struct stationStructure *succStation;
typedef struct packetStructure *packetStruct;
typedef struct routingEntry *routeStruct;


struct stationStructure
{
        int             stationType;
        int             stationRing;
        int             stationAddress;
        int             tokenArrival;
        int             delayStart;
        int             totalStationDelay;
        packetStruct    receivePacket;
        packetStruct    sendPacket;
        succStation     nextStation;
        int             transmitMsg;
        int             receiveMsg;
};


struct networkStucture
{
        int             tokenPosition;
        int             tokenArrival;
        int             stationsOnRing;
        succStation     ring;
        int             totalMsgSent;
        int             totalMsgReceived;
        int             totalRingDelay;
}network[ MAXRINGS ];


struct bridgeStructure
{
        int     bridgeNumber;
```

```
        int     bridge1Ring;
        int     bridge1Address;
        int     bridge2Ring;
        int     bridge2Address;
  bridgeList[ MAXBRIDGES ];


struct routingStructure
{
        routeStruct     routeEntry;
} routingTable[ MAXRINGS ];


struct routingEntry
{
        int             routeRing;
        int             routeBridge;
        routeStruct     nextRoute;
};

struct packetStructure
{
        int             accessControl;
        int             destinationRing;
        int             destinationAddress;
        int             destinationType;
        int             sourceRing;
        int             sourceAddress;
        int             sourceType;
        int             signal;
        int             relevantData;
        int             packetData[PACKETLENGTH];
        int             processId;
        packetStruct    nextPacket;
};


struct onNet
{
        int             packetArrival;
        int             ringDest;
        int             ringAddr;
        packetStruct    packetOnNet;
 onNetwork[ MAXRINGS ];
```

```
/*****************************************************************/
/* Charles Silva                                                 */
/* Master's Thesis                                               */
/* Communications Network for Distributed Real-Time Systems      */
/*****************************************************************/
/* Function to send network topology to a monitor.               */
/*****************************************************************/

#include "tokenring.h"
#include "rptmech.h"

void
MonitorTopology( int *TotalRings )

{

        int                             ringCount=0;
        struct  stationStructure        *current;
        struct  r_data                  req_data;


        /* First node on ring should send and NTTSTART */

        current = network[ ringCount ].ring;

        req_data.msg_id = NTTSTART;
        req_data.msg_type       = 0;
        req_data.value1         = current->stationType;         /* Station Type */
        req_data.value2         = current->stationRing;         /* Station Ring Number */
        req_data.value3         = current->stationAddress;      /* Station Address */
        req_data.value4         = 0 ;                          /* Don't care */
        req_data.value5         = 0 ;                          /* Don't care*/
        strcpy(req_data.msg_text1, "NULL") ;                   /* All don't cares */
        strcpy(req_data.msg_text2, "NULL") ;
        strcpy(req_data.msg_text3, "NULL") ;
        strcpy(req_data.msg_text4, "NULL") ;
        strcpy(req_data.msg_text5, "NULL") ;

        printf("Message Id -> %d      Type -> %d        Ring -> %d       Address -> %d\n",
                        req_data.msg_id, req_data.value1, req_data.value2, req_data.value3);

        current = current->nextStation;


        for( ringCount=0; ringCount <= *TotalRings; ringCount++ )
```

```
{
        if ( ringCount != 0 )
                current = network[ ringCount ].ring;


        while ( current != NULL )
        {

                req_data.msg_id = NTTENTRY;
                req_data.msg_type       = 0;
                req_data.value1         = current->stationType;         /* Station Type */
                req_data.value2         = current->stationRing;         /* Station Ring Number */
                req_data.value3         = current->stationAddress;      /* Station Address */
                req_data.value4         = 0 ;                          /* Don't care */
                req_data.value5         = 0 ;                          /* Don't care*/
                strcpy(req_data.msg_text1, "NULL") ;                   /* All don't cares */
                strcpy(req_data.msg_text2, "NULL") ;
                strcpy(req_data.msg_text3, "NULL") ;
                strcpy(req_data.msg_text4, "NULL") ;
                strcpy(req_data.msg_text5, "NULL") ;


                if (( ringCount == (*TotalRings-1) ) && ( current->nextStation == NULL ))
                {
                        /* Must be at end of network so change to NTTEND and send it */

                        req_data.msg_id = NTTEND;

                        printf("Message Id -> %d      Type -> %d        Ring -> %d       Address -
                                req_data.msg_id, req_data.value1, req_data.value2, req_data.value3


                        break;
                }

                /* Send data to monitor */

                printf("Message Id -> %d      Type -> %d        Ring -> %d       Address -> %d\n",
                        req_data.msg_id, req_data.value1, req_data.value2, req_data.value3);


                /* Pass node address and node type to monitor */


                current = current->nextStation;
```

```
/*************************************************************/
/* Charles Silva                                             */
/* Master's Thesis                                           */
/* Communications Network for Distributed Real-Time Systems  */
/*************************************************************/
/* Function to send report of network statistics to monitor  */
/*************************************************************/

#include "tokenring.h"



void networkReport ()
{

        int     ringNum, networkMsgTrans=0, networkMsgReceived=0, networkDelay=0;

        for ( ringNum = 0; ringNum < numberOfRings; ringNum++ )
        {
                networkMsgTrans += network[ ringNum ].totalMsgSent;
                networkMsgReceived += network[ ringNum ].totalMsgReceived;
                networkDelay += network[ ringNum ].totalRingDelay;
        }


        printf( "Total Messages Transmitted On Network : %d\n",networkMsgTrans );
        printf( "Total Messages Received On Network : %d\n",networkMsgReceived );

        printf( "Average Network Delay : %10.2f\n",(networkDelay / networkMsgTrans));

}
```

```
/*********************************************************/
/* Charles Silva                                         */
/* Master's Thesis                                       */
/* Communications Network for Distributed Real-Time Systems */
/*********************************************************/
/* Function to read in network topology, build network   */
/* structure, build bridge table, instantiate objects, and  */
/* initialize variables                                  */
/*********************************************************/
#include "tokenring.h"
#include "test1.h"

void StuffTopology( char *fileName, int *Nodes, int *Rings )
{
        int index = 0, ringNumber = 0 ;
        int ringAddress = 0, bridgePos = 0;
        int IdType, currentRing = 0;
        char ID[ NODEID ];
        FILE *fileptr ;         /* ptr to file                          */
        struct stationStructure *newStation;
        struct stationStructure *currentNode;
        int foundBridge;


        *Nodes = 0 ;
        *Rings = 0 ;
        bridgeCount = 0;


        /* Attempt to open the file */
        if ( (fileptr=fopen( fileName, "r" ) ) == NULL )
        {
                printf( "Can't open file: %s\n", fileName );
                exit( -1 );
        }

        while (fscanf( fileptr, "%2s\n", ID ) != EOF)
        {


                /* Put ring in structure */

                IdType = atoi( ID );
                if ( IdType == ENDOFRING)
                {
```

```
                ringNumber++;
                ringAddress = 0;
                continue;
        }


        newStation = (struct stationStructure *) malloc(sizeof(struct stationStructure));


        newStation->stationType = IdType;
        newStation->stationRing = ringNumber;
        newStation->stationAddress = ringAddress;
        newStation->receivePacket = (struct packetStructure *) malloc(sizeof(struct packetStructur
        newStation->receivePacket->accessControl = READ;
        newStation->receivePacket->nextPacket = NULL;
        newStation->sendPacket = (struct packetStructure *) malloc(sizeof(struct packetStructure))
        newStation->sendPacket->accessControl = SENT;
        newStation->sendPacket->nextPacket = NULL;
        newStation->nextStation = NULL;
        newStation->transmitMsg = 0;
        newStation->receiveMsg = 0;
        newStation->delayStart = 0;
        newStation->totalStationDelay = 0;

        /* Check to see if station is a bridge. If so, create entry in bridge table */

        if ( IdType == CU )
                printf(" Instantiate a CU \n");

        if ( IdType == ALU )
                printf(" Instantiate a ALU \n");

        if ( IdType == REGISTERS )
                printf(" Instantiate a REGISTER \n");

        if ( IdType == PSW )
                printf(" Instantiate a PSW \n");

        if ( IdType == IOP )
        {
/*              iop iop_test( newStation->receivePacket, newStation->sendPacket ); */
                printf(" Instantiate a IOP \n");
        }
```

```
        if ( IdType == MAINMEM )
                printf(" Instantiate Main Memory \n");

        if ( IdType == SECMEM )
                printf(" Instantiate Secondary Memory \n");

        if ( IdType == COMMPROC )
                printf(" Instantiate a Command Processor \n");

        if ( IdType == OSKERNAL )
                printf(" Instantiate an OS Kernal \n");

/*
        switch( IdType )
        {
                case CU         : printf(" Instantiate a CU \n");       break;
                case ALU        : printf(" Instantiate a ALU \n");      break;
                case REGISTERS  :
                {

                        break;
                }
                case PSW        : break;
                case IOP        : break;
                case MAINMEM    : break;
                case SECMEM     : break;
                case COMMPROC   : break;
                case OSKERNAL   :  break;
                default         : break;
        }

*/

        foundBridge = FALS;
        if (abs(IdType) != IdType)
        {
                for (bridgePos = 0; bridgePos <= bridgeCount; bridgePos++)
                {
                        if (bridgeList[ bridgePos ].bridgeNumber == IdType)
                        {
                                bridgeList[ bridgePos ].bridge2Address = ringAddress;
                                bridgeList[ bridgePos ].bridge2Ring = ringNumber;
                                foundBridge = TRUE;
                        }
                }
                if (foundBridge != TRUE) /* Create a newStation entry */
```

```
                        {
                                bridgeCount++;
                                bridgeList[ bridgeCount ].bridge1Ring = ringNumber;
                                bridgeList[ bridgeCount ].bridge1Address = ringAddress;
                                bridgeList[ bridgeCount ].bridgeNumber = IdType;
                        }
                }



/* Create instantiation of object and pass it pointers to receive and send buffers */
/* Create newStation entry in network geometry table */


                if ( network[ ringNumber ].ring == NULL)
                {
                        network[ ringNumber ].ring = newStation;
                        network[ ringNumber ].stationsOnRing = 0;
                }
                else
                {
                        currentNode = network[ ringNumber ].ring;

                        while (currentNode->nextStation != NULL)
                                currentNode = currentNode->nextStation;

                        currentNode->nextStation = newStation;
                        network[ ringNumber ].stationsOnRing++;
                }

                ringAddress++;
                if ( index > MAXNODES )
                {
                        fprintf( stderr,"Memory overflow error loading topology! \n" );
                        fclose( fileptr ) ;
                        exit( -1 );
                }

        }
        fclose( fileptr ) ;


        for (bridgePos = 1; bridgePos <= bridgeCount; bridgePos++)
        {
                printf("Bridge Number-> %d     1St Address-> %d %d       2nd Address-> %d %d\n",
```

```
                    bridgeList[bridgePos].bridgeNumber,bridgeList[bridgePos].bridge1Ring,
                    bridgeList[bridgePos].bridge1Address,bridgeList[bridgePos].bridge2Ring,
                    bridgeList[bridgePos].bridge2Address);
        }




        /* Initialize data structure which stores which packets are currently on network */


        for (currentRing = 0; currentRing <= ringNumber; currentRing++)
                onNetwork[ currentRing ].packetOnNet = NULL;

        /*Place token at first position on each ring */

        for (currentRing = 0; currentRing <= ringNumber; currentRing++)
        {
                network[ currentRing ].tokenPosition = 0;
                network[ currentRing ].totalMsgSent = 0;
                network[ currentRing ].tokenArrival = 1;
                network[ currentRing ].totalMsgReceived = 0;
                network[ currentRing ].totalRingDelay = 0;
        }


        *Nodes = index ;                /* Number of nodes on entire connected network */
        *Rings = ringNumber ;           /* Number of rings connected to network        */
        numberOfRings = ringNumber;     /* Number of rings on network                  */
}
/*********************************************************/
/* Charles Silva                                         */
/* Master's Thesis                                       */
/* Communications Network for Distributed Real-Time Systems */
/*********************************************************/
/* Function to pass station information to the monitor   */
/*********************************************************/

#include "tokenring.h"


void stationReport (int *stationRing,int *stationAddr )
{
```

```
        struct stationStructure         *currentStation;


        currentStation = network[ *stationRing ].ring;

        while (( currentStation != NULL) &&
               ( currentStation->stationAddress != *stationAddr ))
                currentStation = currentStation->nextStation;


        /* Send to monitor */


        printf( "Total messages transmitted from this station : %d\n",
                currentStation->transmitMsg );
        printf( "Total messages received by this station : %d\n",
                currentStation->receiveMsg );
        printf( "Average station delay : %10.2f\n",
                (currentStation->totalStationDelay / currentStation->transmitMsg));

}
```

```
/*****************************************************************/
/* Charles Silva                                                 */
/* Master's Thesis                                               */
/* Communications Network for Distributed Real-Time Systems      */
/*****************************************************************/
/* Function to check network for any arrivals of messages at a   */
/* station and check for arrivals of a token at a given station  */
/* on a ring.                                                    */
/*****************************************************************/

#include "tokenring.h"
#include "netparms.h"

void findBridgeAddr(int *, int *);
void copyPacket(struct packetStructure *, struct packetStructure *);


void UpdateNetwork(int *simulationTime )

{

        int                             ringCount;
        int                             destAddr;
        int                             destRing;
        int                             packetArrivalTime;
        int                             tokenReturnTime;
        struct stationStructure         *currentStation;
        struct stationStructure         *destinationStation;
        struct packetStructure          *currentPacket;
        struct packetStructure          *freePacket;
        int                             currentBridgeRing;
        int                             currentBridgeAddr;
        int                             packetDestRing;
        struct routingEntry             *currentRoute;
        int                             localDestBridge;
        int                             currBridge;



        /*********************************/
        /* Check for arrival at a station */
        /*********************************/

        for ( ringCount = 0; ringCount <= numberOfRings; ringCount++)
```

```
{
        /*************************************************/
        /* Check for a station wishing to transmit data */
        /*************************************************/


        printf(" Checking for arrivals\n");

        currentStation = network[ ringCount ].ring;

        while ( currentStation != NULL )
        {
                if (( currentStation->sendPacket->accessControl == UNSENT ) &&
                    ( currentStation->delayStart == 0))
                {
                        currentStation->delayStart = *simulationTime;
                }
                currentStation = currentStation->nextStation;
        }



        printf(" Checking for arrivals at desination\n");


        /*************************************************/
        /* Check for arrival of packet at destination */
        /*************************************************/

        if ( onNetwork[ ringCount ].packetArrival == *simulationTime )
        {
                /* An arrival has occurred */

                printf(" Have an arrival\n");
                printf(" Simulation Time = %d    Ring Count = %d\n",*simulationTime,ringCount);



                printf(" onNet ring dest -> %d\n", onNetwork[ ringCount ].ringDest);
                printf(" onNet packet dest -> %d\n", onNetwork[ringCount].packetOnNet->destination


                if (onNetwork[ ringCount ].ringDest != onNetwork[ ringCount ].packetOnNet->destina
                {
```

```
printf(" An arrival has occurred at a bridge\n");

/* Packet going to a Bridge on that ring */

currentBridgeRing = onNetwork[ ringCount ].ringDest;
currentBridgeAddr = onNetwork[ ringCount ].ringAddr;

/* Call function to find bridge's address on other ring */

findBridgeAddr( &currentBridgeRing, &currentBridgeAddr);

currentStation = network[ currentBridgeRing ].ring;

while ( currentStation != NULL )
{
        if (( currentStation->stationRing == currentBridgeRing ) &&
            (currentStation->stationAddress == currentBridgeAddr))
        {

                /* Put packet in bridge's send buffer on other ring and ma
                /** Check to see if there are any packet's currently waiti

                if ( currentStation->sendPacket->accessControl == UNSENT )
                {
                        /* There is a packet waiting to be transmitted, pu

                        currentPacket = currentStation->sendPacket;

                        while (currentPacket->nextPacket != NULL)
                                currentPacket = currentPacket->nextPacket;

                        currentPacket->nextPacket = (struct packetStructur
                                malloc (sizeof(struct packetStructure));
                        currentPacket->nextPacket->nextPacket = NULL;
                        currentPacket = currentPacket->nextPacket;
                        copyPacket(currentPacket, onNetwork[ ringCount ].p
                        currentPacket->accessControl = UNSENT;
                }
                else
                {
                        /* No packets waiting in queue, place at first loc

                        copyPacket(currentStation->sendPacket,
                                onNetwork[ ringCount ].packetOnNet);
```

```
                                        currentStation->sendPacket->accessControl = UNSENT
                                }


                                currentStation->receiveMsg++;
                                network[ currentBridgeRing ].totalMsgReceived++;
                        }
                        currentStation = currentStation->nextStation;
                }
                freePacket = onNetwork[ ringCount ].packetOnNet;
                onNetwork[ ringCount ].packetOnNet = NULL;
                void free(void *freePacket );
        }
        else /* Packet going to a station on that ring */
        {

                printf (" An arrival has occurred at a station\n");

                destRing = onNetwork[ ringCount ].packetOnNet->destinationRing;
                destAddr = onNetwork[ ringCount ].packetOnNet->destinationAddress;
                currentStation = network[ destRing ].ring;
                while ( currentStation != NULL )
                {
                        if (( currentStation->stationRing == destRing ) &&
                        ( currentStation->stationAddress == destAddr ))
                        {

                                /* Give packet to destination station and set access contr

                                copyPacket(currentStation->receivePacket,
                                                onNetwork[ ringCount ].packetOnNet);
                                currentStation->receivePacket->accessControl = UNREAD;
                                currentStation->receiveMsg++;
                                network[ ringCount ].totalMsgReceived++;

                                /* Take packet off network */

                                freePacket = onNetwork[ ringCount ].packetOnNet;
                                onNetwork[ ringCount ].packetOnNet = NULL;
                                void free(void *freePacket );

                        }

                        currentStation = currentStation->nextStation;
                }
```

```
        }
}

/*********************************************/
/* Check for arrival of token at a new station */
/*********************************************/

if ( *simulationTime == network[ ringCount ].tokenArrival )
{

        printf(" Have token at a station\n");

        currentStation = network[ ringCount ].ring;
        while (( currentStation != NULL ) &&
               ( currentStation->stationAddress != network[ ringCount].tokenPosition ))
                currentStation = currentStation->nextStation;


        if (currentStation->sendPacket->accessControl == UNSENT)
        {
                /* Calculate arrival time of outgoing packet, put on network, and mark as

                /* Calculate total delay time of packet */


                printf(" Have packet going to transmit\n");

                network[ ringCount ].totalRingDelay += (*simulationTime - currentStation->
                currentStation->totalStationDelay += (*simulationTime - currentStation->de
                currentStation->delayStart = 0;


                destinationStation = currentStation;

                if (currentStation->sendPacket->destinationRing != ringCount )
                {

                        /* Find path through bridge and put on network */

                        packetDestRing = currentStation->sendPacket->destinationRing;
                        currentRoute = routingTable[ ringCount ].routeEntry;

                        while (( currentRoute->routeRing != packetDestRing ) &&
                               ( currentRoute != NULL ))
                                currentRoute = currentRoute->nextRoute;
```

```
if (( currentRoute->routeRing == packetDestRing ) &&
    ( currentRoute != NULL))
{

        /* Find out which local bridge to route the data to */

        localDestBridge = currentRoute->routeBridge;

}


/* Find bridge on bridge table */


for ( currBridge=1; currBridge <= bridgeCount; currBridge++)
{
        if ( bridgeList[ currBridge ].bridgeNumber == localDestBri
        {

                /* Find bridge address on current ring */

                if ( bridgeList[ currBridge ].bridge1Ring ==
                        currentStation->stationRing )
                {
                        onNetwork[ ringCount ].ringDest =
                                bridgeList[ currBridge ].bridge1Ri
                        onNetwork[ ringCount ].ringAddr =
                                bridgeList[ currBridge ].bridge1Ad
                }
                else
                {

                        onNetwork[ ringCount ].ringDest =
                                bridgeList[ currBridge ].bridge2Ri
                        onNetwork[ ringCount ].ringAddr =
                                bridgeList[ currBridge ].bridge2Ad
                }
        }
}


/* Have bridge's address, Now calculate arrival time and put on ne

packetArrivalTime = *simulationTime;
```

```
while ( destinationStation->stationAddress != onNetwork[ ringCount
{
        packetArrivalTime += STATIONLATENCY;
        packetArrivalTime += NODETONODE;

        if (destinationStation->nextStation == NULL)
        {
                destinationStation = network[ ringCount ].ring;
        }
        else
        {
                destinationStation = destinationStation->nextStati
        }
}


tokenReturnTime = packetArrivalTime;

while ( destinationStation->stationAddress !=
        currentStation->stationAddress )
{

        tokenReturnTime += STATIONLATENCY;
        tokenReturnTime += NODETONODE;

        if (destinationStation->nextStation == NULL)
        {
                destinationStation = network[ ringCount ].ring;
        }
        else
        {
                destinationStation = destinationStation->nextStati
        }
}

/* Put Packet on network and mark it as sent at source station */

onNetwork[ ringCount ].packetArrival = packetArrivalTime;
onNetwork[ ringCount ].packetOnNet = (struct packetStructure *)
        malloc (sizeof(struct packetStructure));
copyPacket(onNetwork[ ringCount ].packetOnNet, currentStation->sen
currentStation->sendPacket->accessControl = SENT;

if (abs(currentStation->stationType) != currentStation->stationTyp
```

```
                {
                        /* Transmitting from a bridge, Check to see if there are a
                        /* in waiting queue.

                        if ( currentStation->sendPacket->nextPacket != NULL )
                        {
                                freePacket = currentStation->sendPacket;
                                currentStation->sendPacket = currentStation->sendP
                                void free(void *freePacket );
                        }
                }


                currentStation->transmitMsg++;
                network[ ringCount ].totalMsgSent++;

                /* Calculate arrival time of token at next station */

                network[ ringCount ].tokenArrival = tokenReturnTime+STATIONLATENCY

                /* Increment token position on ring by 1. If back at first station
                /* give token to first station

                network[ ringCount ].tokenPosition++;
                if ( network[ ringCount ].tokenPosition >
                     network[ ringCount ].stationsOnRing )
                        network[ ringCount ].tokenPosition = 0;


        }
        else
        {

                /* On same ring, Calculate arrival time and put on network */

                packetArrivalTime = *simulationTime;

                while ( destinationStation->stationAddress !=
                        currentStation->sendPacket->destinationAddress )
                {

                        packetArrivalTime += STATIONLATENCY;
                        packetArrivalTime += NODETONODE;

                        if (destinationStation->nextStation == NULL)
```

```
        {
                destinationStation = network[ ringCount ].ring;
        }
        else
        {
                destinationStation = destinationStation->nextStati
        }
}


tokenReturnTime = packetArrivalTime;

while ( destinationStation->stationAddress !=
        currentStation->stationAddress )
{
        tokenReturnTime += STATIONLATENCY;
        tokenReturnTime += NODETONODE;

        if (destinationStation->nextStation == NULL)
        {
                destinationStation = network[ ringCount ].ring;
        }
        else
        {
                destinationStation = destinationStation->nextStati
        }
}

/* Put Packet on network and mark it as sent at source station */

onNetwork[ ringCount ].packetArrival = packetArrivalTime;
onNetwork[ ringCount ].packetOnNet = (struct packetStructure *)
        malloc (sizeof(struct packetStructure));
copyPacket(onNetwork[ ringCount ].packetOnNet, currentStation->sen
currentStation->sendPacket->accessControl = SENT;


if (abs(currentStation->stationType) != currentStation->stationTyp
{
        /* Transmitting from a bridge, Check to see if there are a
        /* in waiting queue.

        if ( currentStation->sendPacket->nextPacket != NULL )
        {
                freePacket = currentStation->sendPacket;
```

```
                                        currentStation->sendPacket = currentStation->sendP
                                        void free(void *freePacket );
                                    }
                                }


                                onNetwork[ ringCount ].ringDest = currentStation->sendPacket->dest
                                onNetwork[ ringCount ].ringAddr = currentStation->sendPacket->dest
                                currentStation->transmitMsg++;
                                network[ ringCount ].totalMsgSent++;

                                /* Calculate arrival time of token at next station */

                                network[ ringCount ].tokenArrival = tokenReturnTime+STATIONLATENCY

                                /* Increment token position on ring by 1. If back at first station
                                /* give token to first station

                                network[ ringCount ].tokenPosition++;
                                if ( network[ ringCount ].tokenPosition >
                                     network[ ringCount ].stationsOnRing )
                                        network[ ringCount ].tokenPosition = 0;

                            }
                        }
                        else     /* Station has nothing to send */
                        {

                            printf(" Have nothing to send going to pass token\n");

                            /* Station has nothing to send, so simply calculate token arrival */
                            /* time at next station                                           */

                            network[ ringCount ].tokenArrival = *simulationTime+STATIONLATENCY+NODETON
                            network[ ringCount ].tokenPosition++;
                            if ( network[ ringCount ].tokenPosition > network[ ringCount ].stationsOnR
                                    network[ ringCount ].tokenPosition = 0;
                        }
                    }

                }

        } /* End function */
```