# Priority Ethernets

Frank Adelstein and Mukesh Singhal
Dept. of Computer and Information Science
The Ohio State University
Columbus, OH 43210-1277

## Abstract

*This paper addresses the issues related to the delivery of multimedia traffic on local area networks. Multimedia integrates voice and video along with text and images into existing systems. Traditional local area networks, e.g., Ethernet, were designed to handle regular data traffic which are bursty in nature and for which variable delay is acceptable. Multimedia traffic, on the other hand, requires a constant delay in addition to fast (or real time) delivery. Multimedia traffic also requires a large amount of bandwidth compared to regular traffic. Multimedia is becoming important in commercial as well as non-commercial organizations and Ethernet is a widely used local area network among most of these organizations; therefore, modification of Ethernet protocols to integrate multimedia streams is an important issue. In this paper, we present solutions to the problems associated with transfer of multimedia data on Ethernet and qualitatively study the performance of these solutions and their suitability to multimedia traffic. Specifically, priority protocols are developed that enable priority transmission of data on Ethernets. Multimedia data can be assigned priority over regular data, and the developed priority protocols can be used to deliver multimedia data over the Ethernet in a timely manner.*

**Key Words:** *Ethernet, multimedia traffic, priority.*

## 1 Introduction and Motivations

Multimedia is becoming important in several domains [1, 2]. By allowing the flexibility of presenting more than one type of information, such as voice, music, still and motion video, in addition to normal text, multimedia computing is entering into office space, businesses, and other professional, non-academic environments.

There are many potential uses of multimedia in the business realm including presentations, lectures, conferences, discussions, and shared work environments. Multimedia provides more information and interaction than that is possible or practical with conventional facilities. Conferences can link people that are too distant to meet face-to-face and provide a better forum for the exchange of ideas than a 3-way telephone call. Shared applications allow multiple people to work on a project.

While the types of multimedia applications are diverse, they share a common underlying trait: "The multimedia data that is being exchanged from node-to-node is real-time in nature and needs to be delivered in a timely fashion. Multimedia data is isochronous, that is, it is sensitive to delays and requires a constant amount of bandwidth, as opposed to bursty traffic." Different media types of course have different tolerances, but they all have deadlines and data past its deadline can not be used [3, 4]. Current approaches of transmitting regular data traffic which is bursty in nature and can tolerate a high variance in delay are not suitable for multimedia traffic which is isochronous [4]. In multimedia systems, intelligent transmission of packets based on a data priority or deadline that will result in a much lower number of discarded packets is required [3, 5].

Most business organizations and academic institutions already have LANs (local area networks) installed. A typical environment consists of workstations or PCs connected by a LAN, typically an Ethernet. Multimedia is getting more and more important to these organizations. As far as the computing power is concerned, the nodes possess sufficient computing resources to support multimedia applications. However, as far as communication is concerned, the current Ethernets do not support high speed multimedia traffic [6, 7, 8]. Current Ethernet protocols do not support real-time, isochronous traffic.

These organizations and institutions have two options for adding multimedia capabilities. First is to attempt to utilize their existing networks by adding some devices and/or software in order to successfully transmit multimedia streams. The other option is use an ATM type network [9, 10]. ATM protocol is designed such that voice and video can be transmitted effectively along with data packets. ATM switches and interfaces will be needed which would add a lot of cost to these organizations. Therefore, a practical solution is to add capabilities to the existing LANs in order to successfully transmit multimedia streams.

## 2 State of the Art

The ISO Reference Model of Open System Interconnection [11] (also known as the ISO 7-layer reference), divides communication into seven different conceptual layers. The highest is the Application Layer, where different programs or applications communicate with each other. The lowest level is the Physical Layer, which is concerned with the actual physical data transmission and the hardware involved. This layers deals with the specifications electrical signals and their timing sent over a physical wire. Above this level is the Data Link Layer or the level 2 layer. This layer specifies how data is sent from one host to another host over a single physical network. It includes the concept of data frames, checksums and some error handling. Ethernet communication falls in this layer. Internet Protocol (IP), Transmission Control Protocol (TCP), as well as most other protocols involved in interprocess communication, represent higher level protocols. When several networks are connected together, IP provides point-to-point delivery of packets [12]. IP addresses are divided into two parts, the network address and the host address. In theory, a maximum of 254 hosts can be attached to a single Ethernet. However, in practice, the number of hosts is much smaller.

Ethernets are a baseband bus communication medium that use CSMA/CD (carrier sense multiple access/collision detect) [13]. CSMA essentially means that a node will listen to the channel before it attempts to transmit, to avoid transmitting while another node is transmitting. If a node finds the channel is currently busy, it waits until the Ethernet becomes free and then begins transmitting. This scheme is known as 1-persistence. A collision occurs when two or more nodes attempt to transmit packets on the Ethernet simultaneously. All packets involved in a collision cannot be used and must be retransmitted. Collision detection allows a node to detect if its packet going out on the bus is being corrupted by another node that started transmitting almost simultaneously. When a collision occurs, each node waits for a random amount of time (called an exponential backoff, due to the mean time of the delay or backoff doubling with every consecutive collision) and then tries again. This paradigm attempts to give all nodes an equal chance to transmit their data, while avoiding subsequent collisions due to repeated retransmission attempts [13, 14]. There is no notion of priority at this level. Thus, transmission of multimedia data on an Ethernet does not get any preference and must compete equally with the transmission of regular data traffic.

Ethernet hardware, i.e., network interface, is designed to minimize the load on the CPU. The interface delivers only those packets to its node that are sent to it or broadcast to the entire net and ignores packets intended for other nodes. Broadcast packets are received by all nodes on an Ethernet, but are not propagated to adjoining Ethernets when the system consists of multiple interconnected Ethernets. The network interface can be configured to be in "promiscuous mode" in which it receives all packets sent on the bus [15]. However, this is likely to cause an unacceptable load on the CPU to process and discard all of the undesired packets.

Thus, Ethernet is designed to efficiently transfer regular data traffic which is bursty in nature and for which variable delay is acceptable (e.g., transaction processing). It is not suitable for multimedia traffic which requires constant delay, fast (or real time) delivery, and a large bandwidth.

Token ring networks support the concept of priority data transfer [16] and suitability of priority data transfer on a token ring to multimedia traffic has been studied by Amer et al. [5]. Their study reveals that the priority mechanism on a token ring considerably improves the quality of multimedia data transfer. Priority Ethernet has been proposed in the context of traditional communication systems [17, 18].

## 3 Priority Protocol for Ethernet

### 3.1 Objectives

Existing Ethernets have two main problems with regard to the integration of multimedia streams. First, the existing bandwidth capability for a standard Ethernets (10Mbps) is not enough to support (future) multimedia transmission requirements. Second, there is no priority mechanism for transmitting data on Ethernets. Therefore, if an Ethernet is used for multimedia, transmission of multimedia data must compete equally with the transmission of regular data. Note, however, that multimedia streams must be received at the presentation node in time such that there is no interruption (or gap) in presentation from user's point of view (consider a moving picture presentation here). That is, the Ethernet must be able to deliver multimedia packets with a small and constant delay (in real time). Clearly, the communication protocols of the Ethernet that are designed to efficiently transfer regular data traffic which are bursty in nature and can tolerate variable delay, must be modified to suit multimedia applications.

In this paper, we develop a set of protocols that enable priority transmission of data on an Ethernet. That is, if a set of nodes are trying to transmit packets, the node with the highest priority packet succeeds. Development of such protocols involves changes to the Data Link protocol layer of the Ethernet to carry out packet transmission in priority order, and changes to higher layer protocols to allow user processes to define, specify, and interpret the priority of data.

Thus, this paper focuses on the problems associated with supporting multimedia on Ethernet environments as well as qualitatively studying the performance aspects of the possible alternatives. By using a protocol for priority transmission, multimedia data can be given a higher priority than regular data and thus allowing multimedia data to be delivered in a timely fashion over the Ethernet.

## 3.2 Issues Involved

There are two main issues involved in the design of priority protocols of an Ethernet, namely, functional decomposition of the protocol and the structuring of the protocol in the layered architecture.

*Functional decomposition of the protocol:* This deals with decomposing the protocol functions in components and identifying what functions the various components of the protocol must perform. There are two distinct functions of the protocol.

1. *Dissemination of state information of nodes.* At any time, the node with the highest priority packet on an Ethernet is allowed to transmit its data on the Ethernet. In order to decide which node has the highest priority packet, each node must have knowledge of the highest priority packet of the other nodes on the Ethernet. Therefore, timely dissemination of this information is an important issue. Each node maintains a queue containing the priority of the highest priority packet every other node wishes to transmit.

2. *Transmission of packets in priority order.* When a node wants to transmit a packet, in addition to the existing CSMA/CD checks, it compares the priority of its packet with that of the other nodes on the Ethernet. This is done by examining the queue containing the priorities. If it has the highest priority, it is allowed to transmit the packet immediately. Ties are broken by an *a priori* ranking of the nodes.

*Structuring issues:* These issues deal with assigning protocols functions to various layers (determining what layers are responsible for what). Priority cannot be implemented at the Network Layer (level 3) or higher layers because implementation of priority at these layers will guarantee enforcement of priority only locally. It will not guarantee transmission of packets in the order of priority at global level (among all nodes). Also, in this case the processing cost and time required to pass information up and down the protocol layers is likely to be high. Therefore, the concept of priority needs to be implemented at the lowest level possible, specifically, the Data Link Layer, which corresponds to the Ethernet device drivers.

Ideally, the network interface, the Ethernet hardware controller, should be "intelligent" and be able to implement a priority protocol on its own (a hardware-oriented solution). However, if that is not possible, priority protocol can be implemented in the device driver controlled by the node's operating system (a software solution). An intelligent interface requires new, sophisticated hardware, and will be more expensive than the existing Ethernet controllers, while a device driver change only requires a software modification. There is a trade-off, however. If priority is implemented in the device driver, the device driver must then interpret all the broadcast priority information packets. This requires interrupting the CPU to process each of these packets. Clearly, the load on the CPU will be much greater in this case than if all the information were handled at the hardware interface level. Device drivers are responsible for writing and reading data to and from the hardware interface buffer. Since the device drivers are interrupt driven, the amount of work they do must be minimized. A context switch is required every time a packet is received and additional time is required to pass information to and from the different protocol layers.

While the basic mechanisms of a priority protocol must be supported at the lowest protocol levels, other parts of it can be located at higher levels. There are costs and benefits with respect to placing the support for the priority protocol at each level and it is these issues are explored in section 4.5.

# 4 Techniques for Priority Transmission

There are two main issues to be resolved: the first is the dissemination of the priority information of the packets at the nodes and the second is the transfer of the packets in priority order. After the priority information of packets at all nodes is disseminated, the rules for transmitting packets on Ethernets in the order of priority are relatively straightforward, i.e., a node waits until its packet has the highest priority. However, dissemination of the priority information of nodes to all the nodes efficiently is a complex issue. Dissemination of priority information should take precedence over the transmission of data on an Ethernet. This is difficult to implement on the Ethernet that does not support priority communication to start with. There are several factors involved that may influence the performance, including the number of nodes, the amount of network traffic, and the distribution of the traffic among various priorities involved.

There are two issues involved in the dissemination of the priority information. The first is transmitting the priority of the highest priority message for every node on an Ethernet. The second is disseminating the information that a node has finished transmitting its data packet or packets so that the node with the next highest priority can start transmission. In this section, we present three methods for disseminating node priority information for the priority protocols on Ethernets.

### Data Structures and Notations

The following data structures are used in all of the three methods. Note that the terms "site" and "node" are used interchangeably. Each node has two priority queues, called *priority* and *pending*. Each element in the *priority* queue at site $i$ represents a site that wishes to send a message, and contains the site's id and the priority of the message it wants to send. Each element in the *pending* queue at site $i$ represents a message that site $i$ wants to send, and contains the priority of the message and a reference to the message. Elements in the queue are stored in priority order, with the elements with the highest priority at the front of the queue. Message priorities can assume three values, *low*, *high*, and *none*, where *low* indicates the site wishes to send a low priority message, *high* indicates the site wishes to send a high priority message and *none* indicates it does not wish to send a message.

All three methods use two types of broadcast messages. The first is *PRIO-REQ*, which requests all sites to broadcast their current priority. The second message, *PRIO-IS*, is used by a site to inform other sites of the priority of its highest priority message and contains a field that stores this priority information. In addition, there is a *YOUR-TURN* message which informs a site that it may now begin transmitting its message. This message is sent (point-to-point, not broadcast) by a site that finishes transmitting its message to the site with the next highest priority message.

Constant $t_{max}$ is twice the maximum time required to propagate a message across an Ethernet.

### 4.1 Synchronous Method

In the synchronous method to distributing the state information, after transmission of an arbitrary number of packets, there is a round of priority information exchange. In this round, each node sends a broadcast packet informing other nodes of the highest priority packet it wishes to transmit. Nodes broadcast their status information in an *a priori* fixed order. For example, node $i$ broadcasts its information after it receives node $i-1$'s state information, after which node $i+1$ broadcasts its state. Provisions need to be made to handle adding and dropping nodes. This method should virtually eliminate Ethernet contention during the priority information exchange phase, however, some channel bandwidth is taken up by exchange of priority information and there is a processing overhead due to these broadcast packets moving up and down the protocol layers.

**Overview:** In this method, each node informs every other nodes of its highest priority message during the priority exchange phase. After that phase, every node has an identical copy of the *priority* queue and the node with the message at the front of the queue transmits its message and removes its entry from its *priority* queue. After having finished its transmission, that node informs the node with the message now at the front of its *priority* queue that it may now transmit its message (via a *YOUR-TURN* message). This continues until there are no entries left in the *priority* queue, at which point a priority exchange phase is initiated. If a node wants to send a message when its *priority* queue is empty, it initiates the priority exchange phase.

**To send a message $M$ from site $i$ to site $j$**
  Add $M$ to the *pending* queue.
  If the *priority* queue is empty, then
    Send *PRIO-REQ* message.

**When a *PRIO-REQ* message is received by $i$ from $j$**
  Delete all entries from the *priority* queue.
  If site $i = 1$ (i.e., $i$ is the first site), then
    Broadcast a *PRIO-IS* message containing the priority of the message at the top of the *pending* queue and remove that message from the *pending* queue or *none* if the queue is empty.

**When a *PRIO-IS* message is received by $i$ from $j$**
  If the priority of the *PRIO-IS* message is not equal to *none* then
    Insert an entry in the *priority* queue for site $j$ with the priority contained in the message.
    If $j = i$ - 1, then
      Broadcast a *PRIO-IS* message containing the priority of the message at the top of the *pending* queue and remove that message from the *pending* queue or *none* if the queue is empty.
    If $j = N$ and the element at the front of the *priority* queue is from site $i$, then
      Transmit the message.
      Remove the entry that corresponds to that message from the *priority* queue.
      Transmit a *YOUR-TURN* message to the site that is at the top of the *priority* queue.

**When a *YOUR-TURN* message is received by $i$ from $j$**
  Remove all elements from the front of the *priority* queue until the message at the front of the queue is from node $i$.
  Transmit the message at the top of the of the *priority* queue.
  Remove the element corresponding to that message from the *priority* queue.
  Transmit a *YOUR-TURN* message to the site that is at the top of the *priority* queue, or transmit a *PRIO-REQ* message if the *priority* queue is empty.

### 4.2 Modified Synchronous Method

The second approach is a variant of the first. The priority information exchange phase occurs after a fixed number of data messages are transmitted. However, rather than having a synchronous exchange of the priority information, nodes on the Ethernet transmit their priority messages in the priority messages exchange phase using CSMA/CD, with exponential backoff to handle collisions.

In addition, in this method if a site does not have any messages to send, then it does not transmit any priority information in the priority exchange phase, unlike the first approach in which a site must explicitly send a message to set its priority to *none*. However, since the number of sites which participate in the information exchange round vary, it is difficult to know when this phase has ended. The priority information exchange phase is considered ended if $t_{TimeOut}$ time units have elapsed since the last successful transmission of priority information. Therefore the duration of the period is dynamically adjusted based on current demand. This makes this approach highly desirable for heterogeneous traffic environments, since nodes that have nothing to send are not required to transmit anything during this phase.

**To send a message $M$ from site $i$ to site $j$**
  Add $M$ to the *pending* queue
  If the *priority* queue is empty, then
    Send *PRIO-REQ* message.
**When a *PRIO-REQ* message is received by $i$ from $j$**
  Delete all entries from the *priority* queue.
  If the *pending* queue is not empty, then
    Broadcast a *PRIO-IS* message containing the priority of the message at the top of the *pending* queue.
**When a *PRIO-IS* message is received by $i$ from $j$**
  Insert an entry in the *priority* queue and remove the corresponding entry from the *pending* queue if $j = i$ for site $j$ with the priority contained in the message.
**If $t_{TimeOut}$ time units have elapsed since the last *PRIO-REQ* message was received at site** $i$ and the element at the front of the *priority* queue is from $i$, then
  Transmit the message.
  Remove the entry that corresponds to that message from the *priority* queue.
  Transmit a *YOUR-TURN* message to the site that is at the top of the *priority* queue.
**When a *YOUR-TURN* message is received received by $i$ from $j$**
  Remove all elements from the front of the *priority* queue until the message at the front of the queue is from node $i$.
  Transmit the message at the top of the of the *priority* queue.
  Remove the element corresponding to that message from the *priority* queue and the *pending* queue.
  Transmit a *YOUR-TURN* message to the site that is at the top of the *priority* queue or send a *PRIO-REQ* message if the *priority* queue is empty.

### 4.3 Asynchronous Method

The third approach is to allow asynchronous broadcast of node priority information. No distinct round of priority information exchange occurs. Instead, the broadcast of priority information is asynchronously embedded in the normal data transmission. That is, the priority data compete with the regular data for network bandwidth using the CSMA/CD protocol to resolve conflicts. In this approach, when a node gets a data packet for transmission from a user level process, it immediately broadcasts the priority of the packet provided the packet has a priority greater than anything pending transmission at that node. The amount of traffic and utilization of the Ethernet by the high and low priority traffic will influence the effectiveness of this method.

**Overview:** In this method, each node informs every other node of its highest priority message asynchronously, when the node wants to send the message. All sites receive the priority information messages since they are broadcast on the Ethernet, so all nodes add the new entry to their *priority* queue. The node with the highest priority message is the first to transmit its message. When it is finished transmitting, it informs the node with the next highest priority (via the *YOUR-TURN* message) that it can begin transmitting its message. When a node receives a *YOUR-TURN* message, it deletes all entries ahead of it in its *priority* queue, since those messages already have been transmitted, and transmits its messages. And so on.

**To send a message $M$ from site $i$ to site $j$**
  Add $M$ to the *pending* queue.
  If there is no higher priority message in the *pending* queue then
    Send *PRIO-IS* message containing the priority of $M$.
    Move $M$ from *pending* queue to *priority* queue.
  If $M$ is at the front of the *priority* queue, then
    Transmit $M$.
    Remove $M$ from the *priority* queue.
    If the *priority* queue is not empty, then

send a *YOUR-TURN* message to the site that is at the top of the *priority* queue.

**When a *PRIO-IS* message is received by $i$ from $j$**

Insert an entry in the *priority* queue for site $j$ with the priority contained in the message.

**When a *YOUR-TURN* message is received by $i$** from $j$ with a value of $k$

Remove all elements from the front of the *priority* queue until the message at the front of the queue is from node $i$.

Transmit the message at the top of the of the *priority* queue.

Remove the element corresponding to that message from the *priority* queue.

Transmit a *YOUR-TURN* message to the site that is at the top of the *priority* queue if the queue is non-empty.

## 4.4 Dissemination of Completed Transmission

Recall that in order for the second highest priority node to begin transmitting, it must be informed when the highest priority node has finished transmitting its packet. We have identified two approaches to disseminating the information that a node has finished transmitting its data. The first approach is to have every node to monitor the Ethernet and update their *priority* queue every time a node successfully transmits data. For current Ethernet interfaces, this requires the hardware to be set in "promiscuous mode" which may put a heavy load on the CPU because it must do context switches and pass the packets up and down the protocol layers. However, if the interface were intelligent and could update the queue without the CPU having to process all the packets, this can be a viable approach.

In the second approach a node, after successfully transmitting data, transmits a packet indicating that it finished transmitting data. New priority information about the node can be included in this packet. This problem has been solved in the previous three methods with the use of the *YOUR-TURN* message. This message is sent point-to-point to the node with the next highest priority message. The other nodes on the Ethernet will not have current information as the the highest priority node, but every since every site starts with an identical copy of the *priority* queue, all sites have the same well defined order of transmission, and know which site should transmit when they complete their transmission. For all three methods, there is an overhead of a packet that is transmitted after every data message. The more packets per data message, the smaller the overhead of the broadcast packets. However, in the worst case when each data message occupies one physical Ethernet packet or frame, the number of messages transmitted on the Ethernet are doubled.

## 4.5 Qualitative Analysis of the Methods

**Efficiency:** The synchronous approach can suffer from delays during the information exchange phase. Since each site must wait until its turn to broadcast its priority information, every delay incurred by each site adds to the time required to complete the entire exchange phase. These delays can be caused by the time it takes the device driver of node's operating system to process the priority information packets before the node can transmit its priority information. In addition sites that have no message to send must still send a *PRIO-IS* message with the priority field set to *none*. In high traffic when most sites have message to send, this method is very efficient.

The second approach alleviates these delays by allowing the network interface to transmit the priority information packets as soon as the Ethernet is free during the priority information exchange phase. However, delays can be incurred due to packet collisions and retransmissions during this phase. In addition, in this method a node must wait for a timeout period before the information exchange phase ends. If the timeout value is too small, then some sites will not be able to transmit their priority information and therefore not be able to transmit their data in the following data transmission phase. If the timeout value is too large, then time is wasted waiting for the information exchange phase to end and network bandwidth utilization suffers. The second method is superior to the first in heterogeneous traffic, where few sites transmit messages at a higher rate than others. If all sites do have a message to send in every phase, then the second method is no more efficient than the first. The number of nodes on the Ethernet, the data traffic patterns and the latency from the device drivers in the first method will affect the relative performance of the first two methods.

For low traffic, the third method is better, since a high priority message does not have to wait for the priority information exchange phase before it can be transmitted. In the other two methods, if a high priority message arrives at a node to be transmitted just after a priority exchange phase, it will have to wait for all of the messages to be transmitted before the its priority information can be exchanged with the other sites. In this regard, the third method is the most fair, however it is also prone to the most collisions, which are discussed subsequently.

**Potential collisions:** Collisions directly relate to wasted bandwidth on an Ethernet, and should be minimized. The first method should be free from collisions since the priority information exchange phase is completely synchronous and only the site with the highest priority message will transmit during the data transmission phase. However, there will be a lower Ethernet bandwidth utilization, since there will be "dead time" on the net during the delays incurred by processing the priority information messages. The only way a collision can occur is if two sites simultaneously send a *PRIO-REQ* message after the system has been idle. The second method should reduce these delays by allowing asynchronous broadcast of the priority information messages during the priority information exchange phase, although delays due to collisions are possible. During actual data transmission, there should be no collisions, since only the highest priority site on the net will attempt to transmit its data. The third method can suffer from more collisions and subsequently delays due to collisions than the other two methods since priority information messages can be broadcast at any time. However, this method will not have the time lost due to the state information exchange rounds that the first two methods have.

**Potential improvements from using a smart interface:** A smart interface is capable of managing the local *priority* queue and updating it appropriately when it receives *PRIO-IS* and *PRIO-REQ* broadcast messages, without intervention from the node's CPU. The first method benefits the most, since the synchronous operations of the priority information exchange phase consume CPU resources. For every *PRIO-IS* message received, each node's CPU must decide if it is its turn to send the node's priority now or not. Every *PRIO-IS* message must go up and down the protocol layers before it can be processed. In addition, each message generates context switches which consume additional CPU cycles. With a smart interface, all the details of when to broadcast its priority information are handled by the smart interface, which would already be tracking which sites have already broadcast their priority information and if it is this node's turn to broadcast its priority now or not.

The other benefit of having a smart interface would be shared by all three methods. That is, time is saved that would be needed to process the broadcast messages, both the time spent for context switches (at least two per message) and the CPU time needed to process the broadcast message itself. This time would be proportional to the number of sites on the local Ethernet, since each site must broadcast its priority.

In addition, the network interface would be able to process the messages sent when a site finishes transmissing its data.

**Fairness:** The priority information exchange phases used by the first two methods can be invoked either after every message, or after some fixed number of messages. This parameter will affect the fairness versus starvation issue of the algorithm. If the value of the parameter is low, then the priority exchange phases occur more often. So it is more likely that a site with a high priority message can send it sooner. However, other sites, with lower priority message might face starvation. If the parameter is larger, then the information exchange phases occur less often and more of the sites will eventually be able to send their messages, however a site with a higher priority message will be forced to wait until after the next information exchange phase until it can transmit its data.

A reasonable compromise is to set the parameter to the number of sites that want to send a message at the priority exchange phase, that is, for the synchronous method the value is the number of sites that did not send a *PRIO-IS* message with a priority values set to *none* and for the modified synchronous method the value is the number of sites that send a *PRIO-IS* message.

**Message overhead:** All three methods should have a comparable message overhead, since one control message is required to establish the priority and one is required to relinquish it, although the collision rate differs for each method. The methods only differ in the way they disseminate that information.

## 5 Conclusions

Traditional protocols for local area networks, e.g., an Ethernet, were designed to handle data traffic which is bursty in nature and for which variable delay is acceptable. These protocols are not suitable for multimedia traffic which requires pretty much a constant delay in packet deliveries as well as fast (or real time) delivery of packets.

In this paper we have presented three protocols for priority transmission of data on Ethernets: (i) the synchronous method, which virtually eliminates packet collisions by having a well-defined order for data transmission that is established during the priority information exchange phase in which nodes have a fixed *a priori* order to transmit their priority data, (ii) the modified synchronous method, which relaxes the ordering constraint during the priority exchange phase to improve efficiency for heterogeneous traffic, and (iii) the asynchronous method, which eliminates the priority information exchange phase to reduce the time high priority messages wait until they can be transmitted. By giving multimedia data priority over regular data, these priority protocols can be used to deliver multimedia data over the Ethernet in a timely manner with low variance in delay. These protocols will allow a higher throughput for multimedia data with fewer packets dropped (due to missed deadlines).

Since multimedia is becoming important in a large number of business, government, and academic organizations and since an Ethernet is a widely used local area network among most of these organizations, priority protocols for Ethernets will enable these organizations to integrate multimedia capabilities into their existing systems with little change. Other alternatives (such as ATM or other high-speed networks) available to these organizations to integrate multimedia capabilities into their systems will cause an expensive hardware overhaul. Besides multimedia applications, these protocols can be used in applications (e.g., control systems, industrial process control, automated manufacturing, air-traffic control, program trading, etc.) where certain tasks must meet deadlines. Critical tasks can be assigned higher priority to ensure better service to them. In addition, these protocols can be used for priority scheduling in a general computing system to give priority to a user because of project deadlines, importance of the user, or because a user pays higher fees for better service, etc.

We identified the functions of new (intelligent) network interfaces that will perform most of the priority transmission related chores in hardware. And finally, we presented a qualitative analysis comparing the three methods of disseminating priority information based on several different criteria.

Future research will involve the following two issues: (i) comparative study of the performance of these solutions and their suitability to multimedia traffic, and (ii) investigating the performance trade-offs and technological demands in placing functions in various layers in the layered architecture (i.e., protocol structuring and placement issues).

## References


[1] A.D. Narasimhalu and S. Christodoulakis. Multimedia Information Systems: The Unfolding Reality (Guest Editors' Introduction). *Computer*, 24(10):6 – 8, October 1991.

[2] Special issue on multimedia systems. *IEEE Journal On Selected Areas In Communicaions*, 1990.

[3] Domenico Ferrari. Client Requirements for Real-Time Communication Services. *IEEE Communications Magazine*, pages 65 – 72, November 1990.

[4] Fouad A. Tobagi. Multimedia: The Challenge Behind the Vision. *Data Communications*, pages 61–66, January 1993.

[5] Khaled Amer, Ken Christiansen, Tora Toher. Experiments With Client/Server Multimedia On Token Ring. In *Proceedings of the 18th Conference on Local Area Networks*, 1993.

[6] Dave Andrews. High Stakes for High-Speed Ethernet. *Byte*, pages 22 – 23, October 1993.

[7] Nick Chinai. A Preview of Fast Ethernet. *SmartNet*, pages 71 – 75, Spring/Summer 1993.

[8] Daniel Minoli. Isochronous Ethernet: Poised for Launch. *Network Computing*, pages 156 – 161, August 1993.

[9] Jean-Pierre Coudreuse. Communications Using ATM. *Encyclopedia of Telecommunications*, 4:123 – 164, 1992.

[10] Jean-Yves Le Boudec. The Asynchronous Transfer Mode: a tutorial. *Computer Networks and ISDN Systems*, 24:279 – 309, 1992.

[11] International Standards Organization. Basic Reference Model for Open System Interconnection. *ISO 7498*, 1984.

[12] Douglas Comer. *Internetworking With TCP/IP: Principles, Protocols and Architecture.* Prentice-Hall, 1988.

[13] R. Metcalfe and D. Boggs. Ethernet: Distributed Packet Switching for Local Computer Networks. *Communications of the ACM*, 19(7):395 – 404, July 1976.

[14] The Institute of Electrical and Electronics Engineers. Carrier Sense Multiple Access with Collision Detections (CSMA/CD) Access Method and Physical Layer Specifications. *American National Standard ANSI/IEEE Std. 802.3-1985.*

[15] Sun Microsystems, Inc. *LANCE 10Mbit/sec Ethernet interface*, June 1992. From the online manual page le(4).

[16] Khaled Amer, Ken Christiansen, Tora Toher. Multimedia Applications on IBM 16-bit Token-Ring LANs. In *IBM Personal Systems Journal*, April 1993.

[17] Herman D. Hughes and U-Fun Io. An ethernet-like protocol with two level priority function. In *Proceedings of the 1988 Summer COmputer Simulation Conference*, July 1988.

[18] Fouad A. Tobagi. Carrier sense multiple access with message-based priority function. *IEEE Tarnsaactions on Communications*, COM-30:185 – 200, January 1982.