Supporting Real-Time Traffic on Ethernet

Tzi-cker Chiueh Chitra Venkatramani

Computer Science Department
State University of New York at Stony Brook
Stony Brook, NY 11794-4400

{chiueh, chitra}@cs.sunysb.edu

March 6, 1994

Abstract

Ethernet is the dominant local area network architecture in the last decade, and we believe that it is going to play the same important role in the future because of its cost-effectiveness and the availability of higher-bandwidth Ethernets. We propose and evaluate a software-based protocol called *RETHER* that provides real-time performance guarantee to multimedia applications *without* modifying existing Ethernet hardware. RETHER features a hybrid mode of operation to reduce the performance impact on non-real-time network packets, a race-condition-free distributed admission control mechanism, and an efficient token-passing scheme that protects the network against token loss due to node failures. We have constructed a detailed simulator to model the behavior of this protocol under various workloads and network configurations. The simulation results and their analysis are presented in this paper, together with a discussion of the practical implementation issues.

1 Introduction

Among the various types of media supported by multimedia information systems, digital audio and video, also known as continuous media, require guaranteed transport bandwidth from the underlying I/O and communication subsystems. Of the two, the communications subsystem plays a more critical role than the I/O component for the following two reasons. First, network resources in modern computer systems are usually shared among multiple agents that are physically distributed. This complicates the issue of guaranteeing network bandwidth because of the distributed decision-making process; whereas in the case of I/O, the allocation decision usually could be made in a centralized fashion. Second, the performance variation of networking tends to be larger than that of I/O. That is, a slight slip in network performance can easily overshadow any performance guarantees provided by the I/O system. Hence, it is much more important to have a tight control over the performance of the network subsystem than that of the I/O subsystem.

Despite the advocacy of high-speed networks such as FDDI and ATM in recent years, the predominant architecture for local area networks is still based on Ethernet. We believe this situation is going to stay the same till the end of this century due to the following technological advancements. First of all, newer generations of Ethernets such as Grand Junction Networks' 100Base-X and HP's 100Base-VG provide 100 Mb/sec bandwidth, which is comparable to the so-called high-speed LANs, at a lower price. Second, the advent of *switch-hubs* has improved the scalability of the Ethernet architecture by allowing the inter-connection of Ethernet segments into a local inter-network. Therefore, even for servicing large-scale organizations, Ethernet still remain a viable choice. Based on these two observations, we believe that Ethernet will continue to be the network of choice in a LAN environment, at least at the leaves of a hierarchy of inter-networks.

Unfortunately, because the medium access protocol of Ethernet allows multiple nodes to compete for access to the channel, it is inherently incapable of guaranteeing bandwidth to any particular node. Although the Ethernet protocol specification has a provision for prioritized access arbitration, this mechanism doesn't in itself offer guaranteed bandwidth to an arbitrary pair of nodes. Besides, most commodity Ethernet controllers do not necessarily implement this feature. Therefore, the major goal of this work is to develop a software-based protocol for existing Ethernet hardware to provide real-time performance guarantees. This protocol is implementable as an enhancement to the existing TCP/IP protocol suite and will work with any off-the-shelf Ethernet hardware and device driver. As will become clear later, the constraint of using commodity networking hardware significantly restricts the range of design possibilities for providing real-time performance guarantees.

The proposed protocol is a combination of CSMA/CD and token passing mechanisms. We will, henceforth, refer to it as the *RETHER* (Real-time ETHERnet) protocol. The basic idea of RETHER is to operate the network with the CSMA/CD protocol when there are no real-time sessions, and to switch it to a token passing medium access control protocol when real-time connection requests do exist. The rationale behind this decision is to provide bandwidth guarantees

to real-time connections while minimizing the performance impacts on non-real-time traffic.

The rest of the paper is organized as follows : Section 2 describes previously proposed approaches to support deterministic network services in a LAN environment. Section 3 describes the RETHER protocol in detail. Section 4 describes the simulation methodology and the data structures used in the simulator. Section 5 presents the simulation results and their detailed analysis. Section 6 discusses the implementation issues involved in modifying a generic UNIX network subsystem to support the RETHER protocol. Section 7 concludes the paper with a summary of main research results from this project and outlines the plan for future work.

2 Related Work

A few different schemes have been proposed to support real-time traffic on the Ethernet. In general, there are two approaches to address the problem of providing bandwidth guarantees on a multiple access communication medium – token passing and synchronous time division multiplexing. The former uses a control packet called the *token* to limit the media accessibility to the token holder, whereas the latter uses absolute time to implicitly synchronize the accesses to the media among the nodes on the network.

An enhanced token bus protocol is described in [CCG93]. In traditional token passing schemes, the token-holding node transmits messages and then passes the token over to its successor in the logical ring. In this new scheme, which they call the "Message/Token" protocol, each node is associated with a unique Medium Access Control ID (MACID) and each message is tagged with the MACID of the current token holder. Since the transmission medium is shared, each node constantly monitors the messages flowing on the medium and determines if it is the next token holder, based on the information in the so-called *local predecessor table*. During initialization, each node is assigned at least one MACID. Multiple MACIDs may be associated with a node to indicate that that node could hold a token that many times in one token-rotation cycle. One of the nodes is elected to be the network manager, whose responsibilities include admission control, token bus organization, initialization and maintenance. A new real-time connection request specifies the required frequency of token visits to the network manager, which then decides whether this request can be serviced according to the current bandwidth allocation.

The first problem with this method is that non-real-time data traffic suffers from the high latency characteristics of a token bus network, even when there are no active real-time (RT) sessions. In addition, the network manager could potentially become the system bottleneck and lead to more message traffic on the network. Also, this scheme requires that every packet be broadcast because the token is piggybacked to the packet. This increases the processing overhead at each node. Lastly, the paper fails to address how a new token-passing schedule is generated on the introduction of a new real-time connection without modifying the schedule of existing real-time connections.

A hybrid token-CSMA/CD protocol for integrated voice/data LANs is proposed in [SSC91],

[GW85]. Two implicit tokens – one for voice and one for data – circulate in the network simultaneously, but with different priorities. The network behaves like a token bus for voice traffic and as a low-priority hybrid token-CSMA/CD channel for data traffic. The channel is slotted with the slot size being an end-to-end transmission delay. All stations are assumed to be synchronized and transmission starts at the beginning of a slot. The voice station in possession of the voice token gets priority to transmit. If it has nothing to transmit, it holds on to the token for a period of time called the token-holding time. During this time, the channel is idle. This is sensed by the data stations, which compete for the channel using the CSMA/CD protocol. One of the data stations possesses the data token and gets a higher priority over others in this phase. When there is a collision on the channel, the node with the token continues to send data until all other nodes are in their backoff phase. It then retransmits the packet without waiting, thereby getting collision-free access to the channel. Once a token-holding node completes its transmission, it implicitly passes the token to its successor. Each node counts the number of busy/idle transitions and determines when its turn is. This avoids explicit control token passing. By suitably adjusting the token-holding time, delay bounds can be guaranteed in this protocol. The problem, though, is that this method has the requirement that all nodes in the network be synchronized. It is also not compatible with off-the-shelf Ethernet hardware.

[JSTS92] describes one of many projects intended to explore what can be done with present-day networks and protocols for multimedia applications like video teleconferencing. The scheme relies on a real-time operating system called YARTOS. The protocol does not provide any reservations/guarantees and is a best-effort strategy. This means that the protocol delivers the highest quality possible given the current load in the network. It has been implemented over the IP layer of the Internet Protocol suite and it dynamically adapts the conference frame rate to the bandwidth available on the network. Two jitter control strategies, both of which compromise quality of service, are described – When there is bursty data traffic on the network, audio/video synchrony is compromised, and when there is continued data traffic on the network, the conference frame rate is compromised.

Many other real-time LAN technologies are coming up recently to support real-time multimedia applications [Dan93]. Among these are IEEE 802.9-based and related isochronous LANs. The IEEE 802.9 Working Group has developed a specification for the support of integrated voice and data in the 10 Mbps range. The standard defines a unified access method that offers integrated voice and data (IVD) services to the desktop for a variety of publicly and privately administered backbone networks, such as FDDI and IEEE 802.x. A variant of this standard is the IsoENET announced by IBM and National Semiconductor Corp. The IsoENET provides 96 out-of-band isochronous 64-Kbps subchannels besides the 10-Mbps Ethernet channel. The Ethernet MAC frames coexist over the same physical medium as the isochronous voice and data, but each has a different format. Also, IsoENET uses regular 10BASE-T star wiring, eliminating the need to recable. But, IsoENET has its drawbacks, the major one being that hubs and PCs have to be fitted with new adapters.

All earlier approaches invariably assume a protocol structure that requires hardware support in the form of either synchronized clocks or intelligent token manipulation which conflicts with our original goal of supporting real-time traffic over commodity Ethernet hardware. In contrast, we adopt a completely software-based approach, requiring changes only to the networking software. The proposed RETHER protocol provides real-time guarantees by allowing the reservation of the requested bandwidth whenever possible, and honoring the reservation at run time via a token passing mechanism. Moreover, by observing the CSMA/CD protocol when there are no real-time sessions, it also reduces the performance impact on non-real-time messages to the minimum. The protocol assumes a decentralized control structure and does not require global synchronization among the network nodes.

3 The RETHER Protocol

Before describing the proposed protocol, we will first outline the constraints under which the RETHER protocol is designed:

- The protocol should be implementable based on existing networking hardware, i.e., off-the-shelf Ethernet cards.
- The protocol should be implementable as an enhancement to existing UNIX-like communication subsystems.
- When there are no real-time sessions, performance should be at least as good as that of the current Ethernet.

Because of the first constraint, the MAC protocol of the Ethernet card, namely CSMA/CD, cannot be changed. But, if collisions are permitted, it would be very difficult to bound the time to put a packet on the network link. This bound is required for real-time performance guarantees. Hence, the RETHER protocol adopts a collision-free approach to regulate the network traffic and is implemented as a software layer immediately above the MAC hardware. The RETHER protocol satisfies the second constraint because it can be readily integrated into existing UNIX network software implementations. The protocol satisfies the third constraint by adhering to the CSMA/CD protocol when there is no real-time session on the network. Consequently, the RETHER protocol has two modes of operation, Real-Time (RT) and Non-Real-Time (NRT), which are described in the following two subsections.

3.1 Non-Real-Time Mode

In the non-real-time mode, the nodes compete for the channel according to the original CSMA/CD protocol of the Ethernet. This protocol is implemented in all the Ethernet adapters available in the market. The protocol is fair, performs well under light loads and deteriorates only as the load

on the network increases. In the RETHER protocol, all nodes remain in this mode until an RT request arrives. The first such RT request causes the network to switch to the real-time mode.

Before starting a real-time connection, the initiating node must issue a reservation request message to declare its intention. In case there is no real-time connection on the network, the initiating node broadcasts a *switch-to-RT-mode* message. All nodes receive this message and set their protocol modes to the real-time mode. They hold off sending data and await the completion of transmission of the packet already in the transmission buffer of the network interface. When the transmission buffer becomes empty, they each send an acknowledgement packet back to the sender of the *switch-to-RT-mode* message. Once the broadcasting node receives the acknowledgements from all the nodes, it generates a token and begins circulating it.

The acknowledgements are crucial to the success of this protocol for two reasons. First, acknowledgements signify the nodes' willingness to switch the network from the non-real-time to the real-time operating mode. Second, the fact that acknowledgements are successfully sent out indicates that the nodes do not have any pending packet in the backoff phase of the CSMA mode. The latter is particularly important to the RETHER protocol because in typical Ethernet cards, the software has no control over the data once the data has been put on the network interface's buffers.

If more than one node generate the *switch-to-RT-mode* message simultaneously, only one of them will be broadcast successfully by the CSMA protocol. On receiving the *switch-to-RT-mode* message, other than the acknowledgement, each node won't send out new packets until it receives the control token. They will reject any *switch-to-RT-mode* request that the local processes may have generated in the meantime and ignore any *switch-to-RT-mode* message that arrives after the first one.

3.2 Real-Time Mode

Because this work is mainly concerned with supporting real-time connections on a local-area network environment, we believe that bandwidth guarantee is much more important than delay and jitter control in this context. Therefore, the RETHER protocol is designed specifically for providing bandwidth guarantee. We use a token passing mechanism to provide such a guarantee because it does not require global synchronization and imposes fewer restrictions on the operating environment. While designing this protocol, we make the following assumptions:

- Ethernet refers to a single Ethernet segment, but not multi-segment networks that span across bridges or routers.

- Each real-time node has to specify the required transmission bandwidth in terms of the amount of data it needs to send during a fixed interval of time, called the *bandwidth allocation interval*. The *bandwidth allocation interval* is chosen to be 1/30 second, which reflects the 30 frames/sec bandwidth need of the target application, video-conferencing.

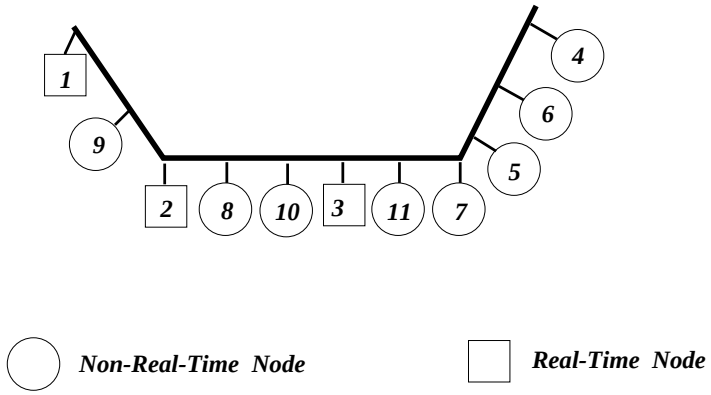

Figure 1: *Example Configuration of RT and NRT Subsets.*

- An average constant bandwidth is reserved for each session. Any bandwidth variation caused by video-compression is absorbed by sufficient buffering at either end.
- There is sufficient buffer space available at the destination node to allow for some jitter in transmission.
- There is only one real-time request per node. This could be trivially generalized to multiple real-time requests per node by treating these requests as virtual nodes that share one physical node.

3.2.1 Basic Algorithm

In the real-time mode, the control token circulates among two sets of nodes with different rotation frequencies – the real-time (RT) set and the non-real-time (NRT) set. The RT set consists of all the nodes that require real-time connections, while the NRT set is comprised of *all* the nodes in the network. That is, a node could receive the control token as an RT node or an NRT node. Figure 1 shows a generic Ethernet segment, where Node 1, 2, and 3 are RT nodes and Node 1, 2, 3,... and 11 belong to the NRT set. Each RT node, once admitted, is guaranteed its reserved bandwidth because the protocol imposes a maximum constraint on the token rotation time and the token holding time for each RT node. In practice, the *Maximum Token Rotation Time* (MTRT) is equal to the bandwidth allocation interval. Part of the total bandwidth is reserved for NRT nodes to prevent starvation. The *Maximum Token Holding Time* ($MTHT_i$) of each RT node is the sum of the required bandwidth divided by the transmission bandwidth, and a fixed software overhead.

When a node in the RT subset receives the control token, it could send either a real-time connection termination message or a unit of real-time data according to the the maximum token holding time. At the end of a real-time token passing cycle, instead of passing the token back to the first RT node, the last RT node, Node 3 in Figure 1 sends it to the nodes in the NRT subset. At this time, the token is tagged with the difference between the MTRT and the sum of the actual token

holding times of the RT nodes in the current token rotation cycle, which represents the residual bandwidth available for NRT nodes. When an NRT node receives the tagged token, it determines if there is sufficient time to send a packet. If so, the node sends the packet, subtracts the tag's remaining time accordingly, and passes the token on to the next NRT node. If not, it notifies the last RT node that in the next token rotation cycle it should be the first NRT node to receive the token, and then passes the token back to the first RT node to start a new token rotation cycle.

For example, suppose the token is passed from the last RT node in Figure 1, Node 3, to the first NRT node, Node 4, to Node 5 and so on. Suppose Node 8 determines that the remaining time on the token is not sufficient to send the first packet on its transmission queue. It then notifies Node 3 that it is the first NRT node to receive the token in the next token rotation cycle, and passes the token to the first RT node, Node 1, essentially restarting another token rotation cycle. In this example, the order through which the token traverses the nodes is as follows: 1 - 2 - 3 - 4 - 5 - 6 - 7 - 1 - 2 - 3 - 8 - 9 ..... Therefore, the token could visit the RT nodes more frequently than the NRT nodes, thus allocating more bandwidth to real-time than to non-real-time traffic.

To ensure that all existing real-time connections not be disturbed by new RT connections, the system performs admission control for every new RT connection request. An important feature of the RETHER protocol is that the admission control decisions are made in a distributed fashion in that every node is responsible for determining whether the remaining network bandwidth is sufficient for the network to admit new real-time connections, or alternatively to convert the node itself into an RT node. To be able to make this decision, every node (RT or NRT) maintains the list of RT nodes and their respective bandwidth requirements. The list is updated every time a node joins or leaves the RT set via a broadcast message.

When a node receives a real-time connection request from a local process with a specified bandwidth reservation, it checks the RT list to see if there is sufficient bandwidth remaining to admit this real-time connection, i.e., converting itself into an RT node. A new connection with a bandwidth reservation $MTHT_{new}$ can be admitted if and only if

$$\sum_{i \in the RT set} MTHT_i + MTHT_{new} + TB_{NRT} \leq MTRT \quad (1)$$

where TB_{NRT} is the minimum amount of bandwidth reserved for the NRT set. If so, it broadcasts its node ID and the bandwidth requirement to all the other nodes, which can then add this information to their RT list.

Because the admission decision is made locally, a race condition could arise in which more than one node try to admit themselves. To resolve this race condition, the admission control decision is made only when the control token is received. In other words, a process' request to initiate a real-time connection will not return until the associated node receives the control token in the NRT mode. Suppose that there is a process in Node 1 and another in Node 3 that simultaneously attempt to establish real-time connections. The node that receives the token first in the NRT mode, say Node 1, gets to perform the admission control test first and broadcasts its reservation

message. All other nodes, after receiving this broadcast message, update their RT lists accordingly. When Node 3 receives the token, it first checks a particular flag that indicates whether there is any local reservation pending. If so, it performs the admission control test with the new RT list that includes Node 1's reservation. Although delaying admission control test to the time when the token is received may seem to prolong the response time of the bandwidth reservation request, this is actually *not* the case since a bandwidth reservation request is valid only when the associated broadcast has been successfully sent out, which is possible only when the node holds the control token.

Because acknowledgements are especially expensive in a token-passing environment, the protocol doesn't require other nodes to acknowledge the reservation broadcast message, unlike the *switch-to-RT-mode* request. The protocol still functions correctly because it is possible to resolve the inconsistency of the RT list due to the anomaly where some nodes did not receive the broadcast message successfully. To guarantee the consistency of RT lists on all nodes, the most up-to-date RT list is included within the token. Given the number of nodes on an Ethernet and the number of RT nodes supportable with 10 Mb/sec, including the RT list in the 64-byte token seems reasonable. After a node successfully joins the RT set, it updates the RT list in the token before passing it on. When a node receives the token, it simply replaces its own RT list with the RT list in the token. Therefore, eventually the RT lists of the network nodes will reach a consistent state. Although there could be a window of temporary inconsistency, this is actually innocuous because the nodes can have an effect on the network only when they hold the control token. But, a node's RT list is guaranteed to be consistent when it holds the control token!

When an RT node wants to terminate its real-time connection, it broadcasts a termination message when it holds the token in the RT mode. All other nodes update their RT lists accordingly, and do not need to return acknowledgements for the same reason described above. When the last RT node decides to terminate its real-time connection, it sends a *switch-to-NRT-mode* broadcast packet asking all nodes to switch back to the CSMA/CD mode. On receiving this message, all nodes switch back to the non-RT mode. The last RT node that terminates its session is also responsible for destroying the token. Again no acknowledgements are needed. However, since there won't be a circulating token to deliver the genuine RT list, in this case an empty set, to every node, it is possible that some nodes may still think they are in the RT mode and wait for the control token when in fact the network is in the NRT mode, i.e., CSMA/CD. To address this problem, the RETHER protocol enforces a maximum interval between successive receipts of the control token in the NRT mode by adjusting TB_{NRT} in Equation (1) appropriately. A node will automatically switch to the NRT mode whenever this timer expires. Consequently, it is not necessary to acknowledge the *switch-to-NRT-mode* message because all the network nodes will eventually converge to the NRT mode even though their RT lists may be temporarily inconsistent.

3.2.2 Reliability Concern

An important reliability issue is when the control token gets lost due to node failure. We solve this problem by making each node responsible for monitoring the state of its logical neighbor. Each node, after sending the token to its neighbor, will set an *acknowledgement timer* whose length is equal to its neighbor's token holding time, and expects to receive an acknowledgement before the timer times out. If its neighbor is an RT node, its token holding time will be equal to its bandwidth reservation period and if it is an NRT node, it will be one MTU time. When the acknowledgement timer indeed times out, it assumes that its neighbor is dead and sends the token along with an updated RT list, if necessary, to the node next to its neighbor.

A potential performance price associated with this token passing scheme is that each token passing transaction requires twice as much time as one without acknowledgement. We address this issue by piggybacking the acknowledgement to a node's previous neighbor with the token passed to its next neighbor. This is possible because Ethernet hardware supports multicast based on the notion of group destination address. Also, because the network operates via a token passing mechanism (no unexpected communication delay), and protocol processing is done by the interrupt handler (no host scheduling problem), the time-out of the acknowledgement timer almost always implies that the monitored node is dead. For example, in Figure 1, Node 1 would monitor Node 2 by awaiting an acknowledgement after sending the token to Node 2. If Node 2 were alive, it would send its message first and then multicast the token to Node 1 and Node 3, its logical predecessor and successor respectively. This token would serve as the acknowledgement without incurring extra delay. If, on the other hand, Node 2 were dead, then Node 1 would time out, and regenerate and send the token to Node 3 with an updated RT list that excludes Node 2.

4 Simulation

The simulator is an event-driven simulator and it simulates the RETHER protocol described in the previous section. The non-real-time workload is characterized by two parameters–the packet size and the packet arrival rate. We adopted the empirical measurement results described in [Ric90] for the packet-size distribution. Although there is no well-accepted model for the inter-arrival time distribution on an Ethernet, we choose a hyperexponential distribution proposed in [SS92] as an approximation. The Cumulative Distribution Function (CDF) for the packet inter-arrival time is

$$F(t) = 1 - (ae^{-t/\alpha_1} + (1-a)e^{-t/\alpha_2}) \tag{2}$$

with a = 0.68, α_1 = 25.2 msec, and α_2=235.2 msec. This model basically assumes that the packet arrival process is a superposition of two Exponential distributions with different mean arrival times.

The time (in μsec) to send a token and to send an MTU (Maximum Transmission Unit) packet were measured on a real Ethernet. These and other constant values used in the simulation were –

```
TOKSENDTIME 680                 /* time to send a token from one node to another*/
```

```
MTUSENDTIME 3450              /* time to send an MTU packet on the Ethernet */
MAXNODES    16                /* number of nodes on the network */
BANDWIDTH   10*1024*1024      /* 10 Mbps Ethernet bandwidth */
```

During the simulation, the token holding time includes the transmission time as well as the associated software overhead that we obtained through measurement. Following is a list of events types processed by the simulator :

```
TOKENARR : Arrival of a token,
RTTERMINATE : Termination of an RT session,
PROTOSW_CT : Protocol switch message (from CSMA to Token Passing),
PROTOSW_TC : Protocol switch message (from Token Passing to CSMA),
PROTOACK : Acknowledgements for PROTOSW_CT,
RTUPDATE : Admission of a new RT session.
```

When each of these events occurs, the corresponding node processes the event as described in the protocol and updates its associated state. The simulator maintains the following state data structure for each node:

```
struct node {
  struct RTnodes *RTlist;          /* list of nodes in the RT set */
  int    RTneighbor;
  int    NRTneighbor;
  int    pendingFlag;              /* TRUE if a local reservation is pending */
  int    protocolMode;             /* current protocol - CSMA or Token Passing */
  struct nonRTmsg *nonRTQ;         /* queue of nonRT requests */
  struct RTmsg *RTQ;               /* queue of RT control messages */
  struct rtstats *RTstats;         /* list of stats of all RT sessions so far */
  struct nonrtstats *nonRTstats;   /* holds stats about nonRT messages */
};

struct RTnodes {
  int    nodeid;                   /* RT node id */
  long   bandwidth;                /* time used in the 33 ms slot */
  struct RTnodes *next;
};
```

RTQ is a queue of RT control messages while nonRTQ is a queue of NRT messages. They are maintained separately because each node may be in the RT set and the NRT set simultaneously. Each node also maintains a linked list of nodes currently in the RT set. Each of the entries in

this linked list indicates the bandwidth reserved for the corresponding real-time session. This data structure is used by each node to make admission decision locally.

The data structure for the control token is

```
struct token {
  char mode;                    /* RT or NRT token ? */
  long residualTimetoDeadline; /* time to deadline of the first RT node */
  int  nextNRTnode;             /* the NRT node to be serviced in next cycle*/
};
```

The mode field indicates if the token is circulating in the RT or NRT set. The residualTimetoDeadline field is used to estimate the amount of time left for NRT message transmission and is updated by the last RT node as well as by the NRT nodes who have received the token. It is only used when the token is circulating in the NRT set. The nextNRTnode field holds the ID of the NRT node which should receive the token when the token is passed from the RT set out to the NRT set the next time around.

Because the RETHER protocol guarantees the required bandwidth for real-time connections once they are admitted, we are mainly interested in investigating the performance impact of this protocol's overhead on NRT messages. The performance metric we use is the average latency of NRT messages. With the above simulator, we were able to estimate the latencies of NRT messages under various workload conditions. The parameters that we varied during the simulations were

- The number of RT sessions
- The bandwidth reserved by each real-time connection
- The NRT load on the network (modeled by the weight 'a' in the CDF of the hyperexponential distribution)
- The maximum token rotation time, MTRT
- The maximum transmission bandwidth of Ethernet

All data used to plot the graphs are averaged over 80 runs, each of which is for a duration of 10 minutes and assuming that TB_{NRT} is zero and the maximum NRT token holding time is an MTU Ethernet packet. The results are presented and analyzed in Section 5.

5 Performance Analysis

This section presents the simulation results and their analysis. Section 5.1 estimates the time to switch between different modes supported in RETHER. Because of the large number of possible combinations of parameter values, we focus on a representative scenario described in Section 5.2, for which the simulation results are analyzed in Section 5.3

5.1 Protocol Mode-Switch Time

It is important to estimate the time the network takes to switch from one protocol mode to the other, because this has a significant effect on the latency experienced by non-real-time messages. The time for the network to switch from the CSMA/CD mode to the Token-Passing mode consists of two components: the time for each node to drain the messages already in the transmission queue of the network interface before the *switch-to-RT-mode* message arrives, and the time for the first RT node to collect the acknowledgements from the remaining nodes. Both components will include the delays due to collisions on the Ethernet. Since it is difficult to simulate the complex CSMA/CD backoff algorithm, we will use an analytical approach to estimate the time it takes for the packet in the transmission buffer to be emptied and the time it takes a node to acknowledge the *switch-to-RT-mode* message.

Suppose that when the *switch-to-RT-mode* broadcast message is received, each node's transmission queue contains an MTU packet, and that the acknowledgement packet is 64-byte long, the minimum Ethernet packet length. When K nodes on an Ethernet want to send packets simultaneously, it has been shown in [And88] that the mean number of contention slots is

$$(1 - \frac{1}{K})^{1-K} \tag{3}$$

and each contention slot is 51.2 μsec. Therefore, for a 16-mode Ethernet segment, the average delays for each non-RT node to drain an MTU packet and send an acknowledgement packet are 3,450+135=3,585 μsec and 680+135 = 815 μsec, respectively.

The time to switch from the Token-Passing to the CSMA mode when the last RT session terminates can be easily estimated since no acknowledgements are needed, as explained in Section 3.2, and there will never be packets in the Ethernet transmission buffer in the Token-Passing mode. When the last RT node broadcasts a *switch-to-NRT-mode* message, it doesn't incur any collision delay, since all the nodes are still in the Token-Passing mode. All nodes, on receiving the protocol switch message, update their states and start operating in the CSMA mode. Hence, the time to switch from the Token-Passing to the CSMA mode is the time required to send out one 64-byte-long *switch-to-NRT-mode* message.

5.2 A Typical Scenario

The following is a description of a typical workload and we will analyze the results for this scenario in more detail.

- The Ethernet consists of 16 nodes.
- In the Token-Passing mode, the MTRT is chosen to be 33.3 msec.
- The real-time sessions are video streams whose spatial resolution is 256x256.

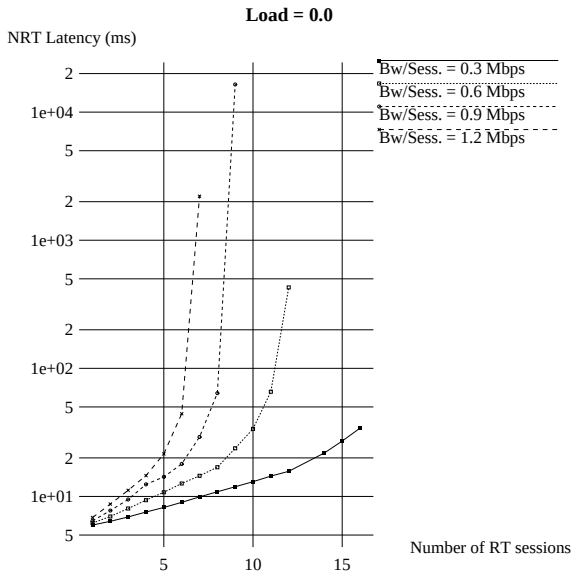

Figure 2: *Number of RT Connections Vs. NRT Message Latency, assuming that each real-time channel reserves a fixed bandwidth, Load = 0.0*

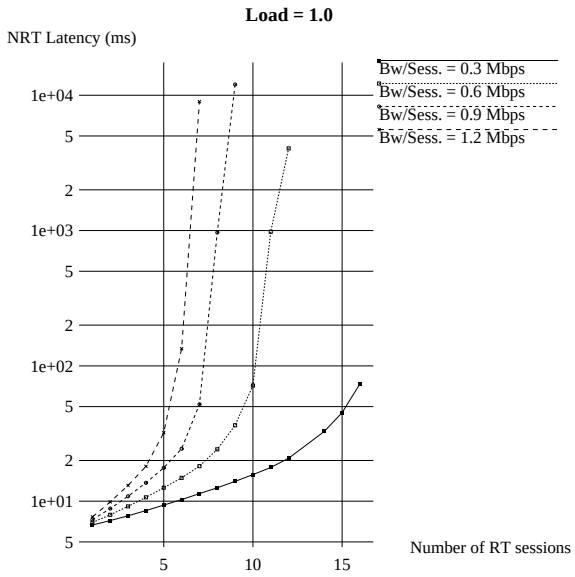

Figure 3: *Number of RT Connections Vs. NRT Message Latency, assuming that each real-time channel reserves a fixed bandwidth, Load = 1.0*

- The uncompressed data bandwidth required per session with 16-bit color and at 30 frames per second would be 30 Mbps. With a compression ratio of 50:1, the amount of bandwidth required per session would be 0.6 Mbps.

When applications reserve the network bandwidth, they have to specify the reservations in the form of the amount of data transmitted within a period of time. There is a difference between 2 MB every two seconds and 1 MB every one sec, although the average data rate is the same. The latter is considered more strict than the former because the former gives more implementation flexibility. To simplify our simulation, we assume that all real-time connections specify their bandwidth needs in the form of data volume transmitted every 33.3 msec.

5.3 Results and Discussion

We are mainly interested in the impact of bandwidth reservation on the performance of the NRT traffic. Hence, all plots have the average NRT message latency in *msec* on the Y-axis in a log scale. The NRT message load on the network is controlled by adjusting the value of 'a' in the hyperexponential distribution for the inter-arrival-times. The higher the value, the shorter the average inter-arrival-time and hence, the heavier the load on the network. In the graphs, the value of 'a' is used to indicate the load on the network.

In Figure 2 and 3, the effect of increasing the number of Real-time sessions is shown. Each plot is for a constant bandwidth per session. Looking at the 0.3 Mbps line, we see that the knee

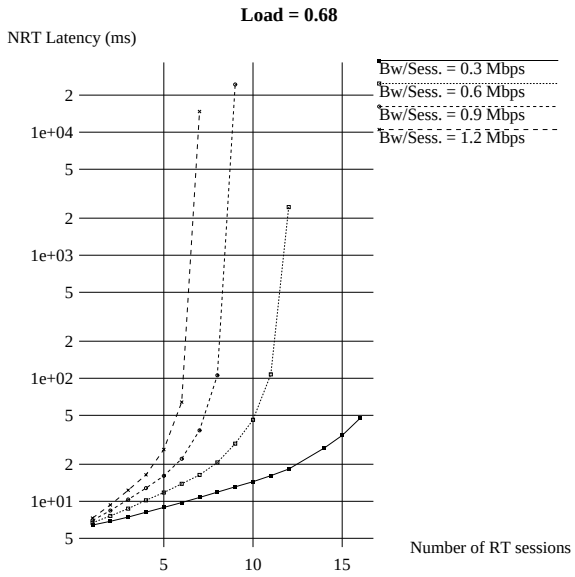

Figure 4: *Number of RT Connections vs. NRT Message Latency, assuming that each real-time channel reserves a fixed bandwidth, Load = 0.68*

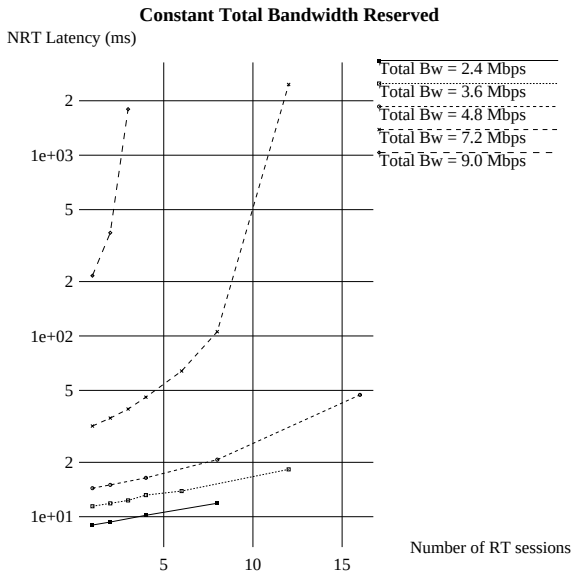

Figure 5: *Number of RT Sessions vs. NRT Message Latency, assuming that the total real-time bandwidth reservation is fixed, Load = 0.68*

of the graph occurs at around 12 sessions and so, if more than 12 such sessions are operating simultaneously, the performance of NRT traffic deteriorates rapidly. Similarly, the other lines represent the performance of the network under different amounts of bandwidth reservation for each real-time session. These two figures show the performance of the system under two extreme distributions of the inter-arrival-times for NRT traffic. As the load increases, it can be seen that the knees occur earlier, which means that fewer RT sessions can be supported without seriously hurting NRT message latencies. Based on these graphs, one can determine the maximum number of real-time sessions that can be supported while maintaining a reasonable performance level for NRT traffic, taking into account both the real-time bandwidth reservation and the NRT traffic load. Figure 4 also plots the latency versus the number of real-time sessions supported. The load in this case is 0.68, which is used in [SS92] to model the generic workload on the Ethernet.

It is interesting to compare the NRT message latencies under a fixed total real-time bandwidth reservation but with different numbers of real-time connections. This would show the effect of the protocol overhead, namely token passing overhead, as the number of RT sessions increases. For example, in all of the three figures above, the NRT message latency for the "10-connection, each 0.3 Mbps" case is always greater than the "5-connection, each 0.6 Mbps" case. The difference is due to the token passing overhead and can be used as a measure of the price of providing real-time performance. This is demonstrated in Figure 5 which is a plot of the latencies against the number of RT sessions for different total bandwidths reserved. All plots have a positive slope indicating

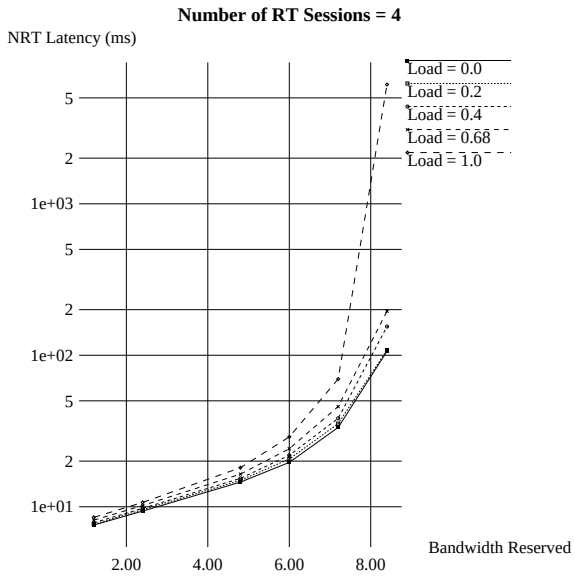

Figure 6: Total Real-Time Bandwidth Reservation vs. NRT Message Latency, assuming a fixed number of real-time channels

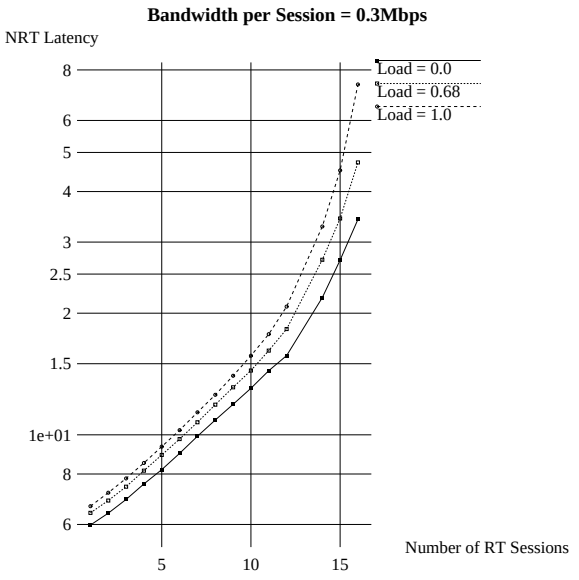

Figure 7: Number of RT Connections vs. NRT Message Latency, assuming a fixed bandwidth reservation for each real-time channel

that the latency increases with the number of RT sessions for the same total bandwidth reserved. It can also be seen that the latency increases gradually with the number of RT sessions when small fractions of the Ethernet bandwidth are reserved. However, the effect of the number of RT sessions and hence, the protocol overhead, becomes more significant once larger fractions of the bandwidth begin to be reserved. Again, 6 Mbps seems to be the threshold fraction of bandwidth for real-time reservation before the protocol overhead starts to render the curve with a larger slope.

To get a better understanding of the global allocation of NRT and RT bandwidth, Figure 6 shows the effect of increasing the amount of real-time bandwidth reserved on the latencies for different NRT loads. The number of RT sessions here is four and so the protocol overhead due to token-passing is fixed. The X-axis shows the total amount of network bandwidth reserved for real-time traffic. For our typical scenario, the effect of reserving 1.2 Mbps does not affect the latency significantly event if the load on the Ethernet is increased. However, for the worst case, where around 8.5 Mbps are reserved, the deterioration with load is significant. It can be concluded from this graph that up to a maximum of 6 Mbps can be safely reserved without degrading the performance of NRT messages significantly. But, this conclusion is only valid for 4 RT sessions since more RT sessions under the same total real-time bandwidth will incur extra token passing overhead.

Next, we measure the effect of the number of RT sessions (with each session reserving 0.3 Mbps as in our typical scenario) on the NRT latencies for different NRT loads. Figure 7 shows the result

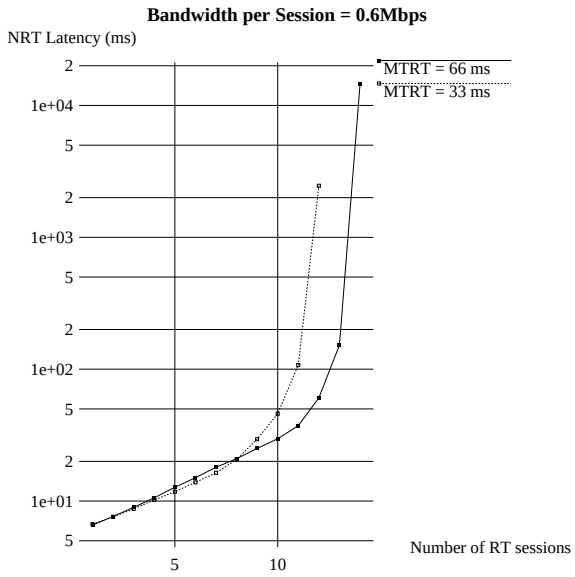

Figure 8: Number of RT Connections vs. NRT Message Latency, MTRT = 66 msec or 33 msec, Load = 0.68

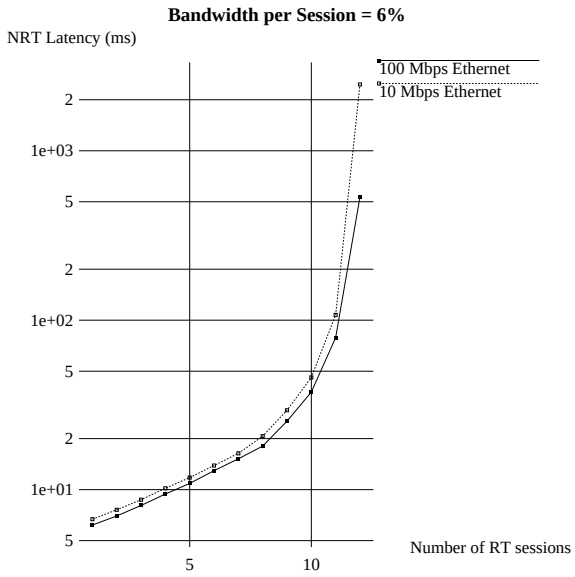

Figure 9: Number of RT Connections vs. NRT Message Latency, transmission bandwidth is 100 Mb/sec or 10 Mb/sec, Load = 0.68

of this experiment. It seems that the difference among various NRT loads becomes significant only when the amount of bandwidth reserved for real-time traffic becomes large.

We also studied the effect of increasing the MTRT to 66 msec and doubling the reserved bandwidth in each real-time connection. Although the total real-time bandwidth requirements remain the same, a larger MTRT reduces the token passing overhead since each RT node can transmit twice as much data within a token rotation cycle. On the other hand, the NRT nodes receive the token at later times than in the 33-ms MTRT case. That is, in the 66-ms MTRT case, the NRT nodes have to wait a longer time on an average to receive the token. This effect is more significant when small fractions of the total bandwidth are reserved. Only when the number of RT sessions increases beyond a certain extent does the reduced token overhead make the 66-ms MTRT case perform better. This effect is shown in Figure 8, where the latency of NRT messages in the 66-ms MTRT case are indeed reduced as compared to the 33-ms MTRT case only when there are 8 or more RT sessions. Below 8 RT sessions, the 33-ms MTRT case exhibits slightly smaller NRT message delay because of faster toke rotation cycle. However, as explained earlier, larger MTRT's correspond to less stringent real-time bandwidth requirements. This graph, therefore, also demonstrates the fundamental tradeoff between RT sessions' real-time resolution requirements and the latency experienced by NRT messages.

Lastly, we studied the effect of using a 100 Mbps Ethernet. Increasing network bandwidth only reduces the raw transmission time of network packets but will not affect the software overhead for

message transmission. Therefore it would only slightly reduce the transmission time of a token or an NRT packet since it is known that software overhead constitutes a significant percentage of the overall message delay. The simulations were run by reducing the token and NRT message passing time accordingly, and assuming that each real-time session consumes 6% of the raw network bandwidth. The result is shown in Figure 9. Compared to the 10 Mbps Ethernet, the knee of this graph has shifted to the right, which indicates that a greater number of RT sessions can be supported on a 100 Mbps Ethernet for the same percentage of total real-time bandwidth reservation. However, the shift is relatively small, which confirms the dominance of the software overhead.

6 Implementation Issues

A generic TCP/IP-based networking subsystem such as the one in BSD UNIX [SMMJ88] consists of three software layers: Transport (TCP/UDP), Network (IP), and a device-dependent interface software (IF). To implement the proposed protocol without hardware support, both the IP and IF layers need to be modified. Also, a UDP-like transport layer would also have to be developed to provide the transport service. The RETHER protocol adds six types of control packets:

```
Token
Acknowledgement
Switch_To_RT_Mode
Switch_To_NRT_Mode
Join_the_RT_Set
Leave_the_RT_Set
```

The first two types are point-to-point message transfers and the remaining four require broadcasting, which could be readily supported by existing Ethernet hardware. These packets requires very simple processing and could be handled directly in the interrupt handlers triggered by the network interface, without the involvement of higher-layer software. To do this, these control packets should be formatted as special Ethernet packets with appropriate Ethernet headers to be dispatched to the appropriate handling routine.

```
ETHERTYPE_TOK
ETHERTYPE_ACK
ETHERTYPE_SW
ETHERTYPE_RT
```

are the four types of Ethernet packets, where the two protocol switch packets and the two RT Set packets are both integrated in one routine. To support real-time communications, high-level networking software must provide interfaces for applications to reserve the bandwidth needed for these real-time connections. With the bandwidth reservation mechanism in place, there is no need for flow control. Moreover, according to our study, current network hardware is so reliable that the

probability of packet loss due to random errors is very unlikely. In the case of real-time data streams, retransmitted packets probably are not useful most of the time anyway. Therefore, we believe a simpler transport layer interface and implementation such as UDP is actually more appropriate for real-time connections. In addition, the IF and IP layer should implement two separate transmission queues for RT and NRT messages in case the node is in the RT and NRT set at the same time. While in the Token-Passing mode, the IF software needs to check whether the node currently holds the token. If not, the data packet to be transmitted must be delayed until the token arrives.

When the protocol has to switch from the CSMA to the Token-Passing mode, the interrupt handler corresponding to this type of packet should set the protocol mode flag, which indicates that the network is no longer operating in the CSMA mode. The node then queues an acknowledgement packet as the next message to be sent out. The Ethernet driver finishes the packet currently in the middle of transmission, if any, and then schedules the acknowledgement packet as the next packet for transmission. Any other messages will be sent out only on receipt of the control token or after another protocol switch message arrives. In summary, all the required modifications to support the RETHER protocol can be made at the lowest layers of the network subsystem. Most of these modifications are rather simple. The higher software layers remain largely unaffected except that they should provide mechanisms to access the features of the RETHER protocol, in particular, bandwidth reservation.

7 Conclusion

The main thrust of this work is to develop a protocol that could provide performance guarantee to real-time connections on top of existing Ethernet hardware. Constrained by off-the-shelf Ethernet interfaces, we design a token-passing protocol that requires very simple processing and thus incurs little protocol overhead when run by the host software. The RETHER protocol adopts a token passing mechanism to guarantee real-time performance requirements and switches to the hardware-supported CSMA/CD protocol when no such real-time connections exist. The rationale of this hybrid approach is to minimize the impact of real-time connections on non-real-time traffic. One of the difficulties in providing bandwidth reservation is that current Ethernet network hardware doesn't provide software a mechanism to cancel packets already sent to the network interface. As a result, when the network is switching to the Token-Passing mode, explicit acknowledgements from other network nodes are needed to signify both their willingness to cooperate and that the interface's transmission buffer is emptied. For other types of control messages, mechanisms are developed to ensure that no acknowledgements be needed, thus significantly reducing the associated overhead.

We use a simulation approach to study the behavior of the RETHER protocol. Whenever possible, the parameter values to the simulator are measured on a real network. The simulation is focused on the token passing mode of the protocol. From the simulation results, the first conclusion is that it is possible to use the proposed RETHER protocol to support multiple real-time sessions

on Ethernet. For real-time sessions that required 0.3 Mbps in average, it is possible to sustain more than 14 sessions without seriously affecting the performance of NRT traffic. Also, from the simulations, it can be concluded that given the assumed NRT workload, up to 60% of the 10Mbps bandwidth of the Ethernet can be reserved using the protocol described, without seriously affecting the NRT traffic. This conclusion holds regardless of the number of real-time connections.

Because the RETHER protocol is designed for a local area network setting, we believe bandwidth guarantee is much more important than jitter and delay guarantees. However, to extend this protocol to a local internetwork, i.e., one that extends across routers/gateways, these issues need to be resolved. Currently, we are in the process of implementing the RETHER protocol in the network subsystem of the BSD UNIX platform. Once the implementation is completed, actual performance measurements will be compared to the results presented here to validate the simulation model. We expect to include these results in the final form of this paper. In the future, we plan to extend this work in the following ways:

- Extension of the protocol to support multiple real-time connection requests per node
- Extension of the protocol to run across bridges and routers
- Enhancement of the protocol to utilize multiple Ethernet links simultaneously as a cost-effective way of supporting multimedia applications
- A more general treatment of the reliability problem associated with nodes and links

References

[And88] Andrew S.Tanenbaum. *Computer Networks.* Prentice-Hall Inc., 1988.

[CCG93] Cheng, Ting, Chung, Jen-Yao, and Georgiou, Christos J. Enhancement of Token Bus Protocols for Multimedia Applications. *Digest of Papers. COMPCON SPRING 1993*, pages 30–36, February 1993.

[Dan93] Daniel Minoli. Isochronous Ethernet : Poised for Launch. *NETWORK COMPUTING*, pages 156–162, August 1993.

[GW85] Gopal, P.M. and Wong, J.W. Analysis of Hybrid Token-CSMA/CD Protocol for Bus Networks. *Computer Networks & ISDN Syst.*, pages 131–141, September 1985.

[JSTS92] Jeffay, K., Stone, D.L., Talley,T., and Smith, F.D. Adaptive, Best-Effort Delivery of Digital Audio and Video Across Packet-Switched Networks. *Network and Operating Systems Support for Digital Audio and Video*, pages 3–14, November 1992.

[Ric90] Riccardo Gusella. A measurement Study of Diskless Workstation Traffic on an Ethernet. *IEEE Transactions on Communications*, 38(9):1557–1567, September 1990.

[SMMJ88] Samuel J.Leffler, Marshall K.McKusick, Michael J.Karels, and John S.Quarterman. *The Design and Implementation of the 4.3 BSD UNIX Operating System.* Addison-Wesley Publishing Company, Inc., 1988.

[SS92] S O Falaki and S-A Sorensen. Traffic measurement on a local area computer network. *Computer Communications*, 15(3), April 1992.

[SSC91] Shieh, Meng-Tsong, Sheu, Jang-Ping, and Chen, Wen-Tsuen. Decentralized token-CSMA/CD protocol for integrated voice/data LANs. *Computer Communications*, pages 223–230, May 1991.