RETHER: A Software-Only Real-Time Ethernet for PLC Networks

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

chiueh@cs.sunysb.edu http://www.ecsl.cs.sunysb.edu/~chiueh

Abstract

The networking technologies used in industrial automation are required to support real-time performance guarantees to ensure that sensor/command data reach target nodes within a delay bound. With the continuing popularity and thus the accompanied price drop of the Ethernet technology, replacing the typically closed and proprietary automation networks with Ethernet is emerging as a very attractive solution, because of the cost-effectiveness and its compatibility with the trend of moving industrial control systems to PC-based hardware. Due to its contention-based media access control protocol, Ethernet theoretically can not bound the network access delay without additional traffic control mechanisms. In this work, we present the design, implementation, and programming interface of a software-based traffic control protocol called *RETHER* , which turns off-the-shelf Ethernets into real-time networks without any hardware modifications. *RETHER* works both in a single-segment as well as a multi-segment environment. This paper reports the performance measurements and implementation experiences with the *RETHER* prototype, which has been fully operational for over 12 months.

1 Introduction

The basic building blocks in industrial automation systems are programmable logic controllers (PLC), dedicated control devices that interface with the physical instruments or sensors. Traditionally PLCs are connected together and with central control computers through a closed and proprietary network technology, because of special timing and/or hardware constraints. As Ethernet dominates the desktop computer world, its price is dropping precipitously. For example, 10 Mbits/sec ISA-based Ethernet adapters cost about $15, and yet 10 Mbits/sec is abundant for automation control applications. Moreover, as PLCs are themselves evolving towards PC-based hardware, choosing Ethernet as the backplane network technology makes even more sense.

The only problem is that commodity Ethernet hardware can not provide any performance guarantee, for the following two reasons. First, Ethernet's media access control protocol is CSMA/CD, which relies on an exponential backoff algorithm to resolve link collision among multiple nodes when they attempt to send data simultaneously. Due to the probabilistic nature of the exponential backoff algorithm, network access delay is inherently non-deterministic. Second, Ethernet does not support prioritization of packets. This means that time-critical network packets could be held up waiting for time-insensitive packets.

The *RETHER* project set out to develop an efficient delay/bandwidth bandwidth guarantee mechanism over off-the-shelf Ethernet without any hardware modification. With *RETHER* , industrial automation systems could use commodity Ethernet as the underlying network, thus reaping the benefits of economies of scale from the PC industry. *RETHER* is designed to be a software-only solution that is built into the device driver of the host operating system. Because it is part of the device driver, *RETHER* is transparent to higher-level network protocols such as TCP/UDP and IP. Consequently, all existing network applications can continue to run on a *RETHER* network without any changes. New real-time applications have to be specifically written against the API provided by *RETHER* , which is based on the industry-standard socket interface.

The design of *RETHER* also minimizes the performance overhead associated with supporting delay/bandwidth guarantees. In particular, *RETHER* features a hybrid mode of operation that automatically switches an Ethernet network between the *RETHER* mode and the CSMA/CD mode, depending on whether there are real-time connections active at the time. In the case of no active real-time connections, the network operates according to the CSMA/CD protocol, thus providing the same performance to non-real-time applications. To avoid starvation, *RETHER* ensures that a certain amount of bandwidth be reserved for non-real-time traffic. The reserved amount is chosen so that existing higher level network protocols such as TCP or NFS would not time out unnecessarily, and thus are to a large extent isolated from the existence of *RETHER* .

The rest of this paper is organized as follows. In Section 2, we describe the programming interface and performance guarantee mechanisms of *RETHER* in the single-segment Ethernet environment. Then we use the single-segment *RETHER* protocol as the building block for multi-segment *RETHER* in Section 3. In Section 4, we present the performance measurements and implementation experiences of *RETHER* from our operational prototype. Section 5 reviews related work in this area to put the contribution of *RETHER* in perspective. Section 6 concludes this paper with a summary of the *RETHER* project and the current status of the report.

2 Single-Segment RETHER

2.1 Overview

RETHER provides a set of procedural interfaces for applications to reserve network bandwidth, and guarantees the reservations throughout the lifetime of the applications once they are admitted. Currently *RETHER* is implemented within the Ethernet device driver under FreeBSD UNIX, Linux and DOS. Under *RETHER* , individual applications can send a certain amount of data (D) in each periodic cycle, T. The amount of data is specified in the reservations and thus varies from application to application. The length of the cycle is fixed but can be changed at the system initialization time. To start a *RETHER* connection, which is *unidirectional*, the sender application makes a library call, `reservation()`, which, upon successfully creating a *RETHER* connection, returns a socket descriptor. From that time on, the sender application can `send()` the reserved amount of data through the returned socket descriptor, and *RETHER* ensures that the data be delivered at the rate $\frac{D}{T}$. The receiver application simply receives the sent data using normal `receive()` calls without making any special arrangements.

The current implementation of `reservation()` is based on the socket interface and uses a special unused Ethernet address[1] to alert the local *RETHER* module in the device driver of the reservation request. Upon detecting such a request, *RETHER* reserves a *RETHER* connection to the target receiver according to the requested bandwidth requirement, and if successful, records a mapping between the reserved *RETHER* connection's ID and the source port number associated with the socket descriptor to be returned to the sender application. At run time, *RETHER* consults with this mapping to determine which *RETHER* connection each IP packet should be sent on. Packets that are not parts of reserved connections are sent on a pre-established *RETHER* connection by default.

In the absence of real-time connections, the *RETHER* network operates under the original CSMA/CD protocol. When the first real-time bandwidth reservation request arrives, the *RETHER* network switches to a *token-passing* mode. In this mode, channel access for *all* traffic, real-time and non-real-time, is regulated by a token. Intuitively time is divided into cycles. In each cycle, the token first services all the nodes that have real-time data to send (called *real-time (RT)* nodes), and then it attempts to service *non-real-time (NRT)* nodes in a round-robin fashion. Essentially NRT nodes share the bandwidth remaining in each cycle after all RT nodes have been serviced. Note that *all* network nodes, including those making bandwidth reservations, are NRT nodes. The network stays in the token-passing mode until the

[1] `reservation()` itself uses a special IP address, which is mapped to the special Ethernet address through an artificial and persistent ARP cache entry.

last real-time session terminates. At this time, the network switches back to the CSMA/CD protocol. This hybrid scheme reduces the performance impact due to token passing on non-real-time traffic.

2.2 Token Passing

The token circulates in cycles so that real-time connections can access the network periodically. Because *RETHER* does not assume a globally synchronized clock, the token cycle time is maintained as a counter called the *residual cycle time* in the token itself. At the beginning of each cycle, the *residual cycle time* is set equal to a full token cycle time[2]. When the token visits a node, the node subtracts its token-holding time from this counter. Once the residual cycle time reaches zero[3], the token is passed back to the first real-time node and a new cycle begins. Figure 1 shows an example token visit schedule within a token cycle.

The token-holding time at each node is based on the amount of data the node is allowed to send. For real-time nodes, the bandwidth reservation determines the amount of data that needs to be sent out during every cycle. For non-real-time nodes, the amount of data that can be sent out is limited by the the total unreserved bandwidth and the size of messages in their output queues. *RETHER* has incorporated a mechanism to ensure that all the nodes use the unreserved bandwidth fairly. When the token visits a node, if it does not have data to send, it merely hands the token to its successor after subtracting the time to process the token.

2.3 Fault Tolerance

As the control token represents a single point of failure, *RETHER* incorporates a built-in fault tolerance mechanism to ensure continued network operation despite token loss due to machine failures, or token corruption due to random bit errors. Each *RETHER* node is required to monitor the health of its successor in the token passing schedule. When a node N sends the token to its successor S, it starts an acknowledgment timer waiting to hear from S. If S is alive, it sends an acknowledgment back to N when it sends the token forward to its own successor. If the successor node is dead for some reasons, the timer at the monitoring node times out and it pings the successor to ensure that the successor indeed dies. This extra ping is necessary to check if the successor is still alive but actually drops the token due to reasons like bit errors. On detecting a failure, the monitoring node broadcasts a message announcing the failure and regenerates a new token. The choice of the timeout value and other failure scenarios are discussed in greater detail in [10]. Although token recovery can take place within the same token cycle, in practice, it takes a little bit longer than one token cycle to recover. This is dependent on the precision with which we can set the acknowledgment timer value in the operating system[4]. *RETHER* also addresses many other failure scenarios such as multiple node failures in [10]. When a new node boots up, it broadcasts a message identifying itself. The node currently holding the token adds the new node to the list of live nodes maintained in the token. As a result the token will visit the new node in the next cycle.

2.4 Admission Control

RETHER incorporates a distributed admission control scheme that is guaranteed to be free of race conditions. The admission decision at a node is postponed until it receives the token, because the token carries the most up-to-date information about all active real-time and their bandwidth reservations. Since only one node holds the token at a time, mutual exclusion is automatic. A disadvantage with this scheme is that bandwidth reservation requests are delayed due to the waiting for the token to arrive before the admission decision can be made.

RETHER intends to support both real-time and non-real-time traffic on the same Ethernet segment. To prevent starvation for non-real-time traffic, only a fixed fraction of the total raw bandwidth is set aside for real-time connections. The reserved bandwidth for non-real-time traffic minimizes unnecessary timeouts for existing network protocols such as NFS, and places an upper bound on the token inter-

[2] This is a system initialization parameter and can be, for instance, set to 33 msec for applications requiring 30 video frames per second.

[3] In practice, the token is passed back to the first RT node when a non-real-time node determines that the residual cycle time in the counter is less than that needed to send out the packet at the head of its message queue. Therefore, the token cycle counter need not be zero when a new token cycle is started.

[4] Most UNIX systems' timer resolution is 10 msec.

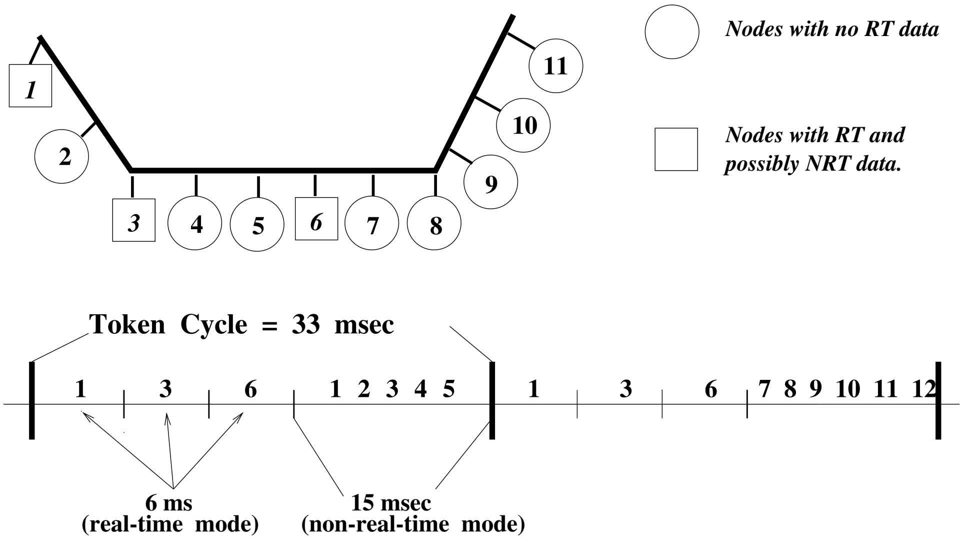

Figure 1: *Node 1, 3, and 6 are real-time nodes, each of which is assumed to have a token holding time of 6 msec every time the token visits them in the real-time mode. As every network node has non-real-time data to send, ALL network nodes are non-real-time nodes. In this example, the token visits real-time nodes in the first 18 msec of the 33-msec token cycle, and then visits non-real-time nodes in the rest of the cycle in a round robin fashion. Note that the token continues the visiting schedule for non-real-time nodes in the next cycle (Node 6) from where it left off in the previous cycle (Node 5) .*

arrival time as seen by NRT nodes. This bound is critical to a timer-based mechanism for detecting and recovering from disastrous failure scenarios.

3 Multi-Segment RETHER

Due to electrical signal considerations, multiple Ethernet segments connected by bridges or switches are required to accommodate a large number of PLC nodes. To provide end-to-end network bandwidth guarantees between any pair of nodes in a multi-segment Ethernet environment, *RETHER* has been extended to operate across switches. Conceptually, a real-time connection applies the single-segment *RETHER* protocol to reserve on each of the segments on the path from its source to the destination. The per-segment reservations along the way are parts of one logical real-time connection. Consider the network configuration in Figure 2. Suppose a real-time connection runs from A1 to C1. This connection is broken down into three sub-connections namely A1-Gw1 on Segment 1, Gw1-Gw2 on Segment 2, and Gw2-C1 on Segment 3, where the nodes on the left are the sender and those on the right are the receivers. For a multi-segment real-time connection, each intermediate switch acts as a sender on one segment and a receiver on the other. All Ethernet segments run the single-segment *RETHER* protocol, with the token cycles on different segments proceeding completely independently of one another. That is, token cycles associated with adjoining segments, although of the same length, are not synchronized at all.

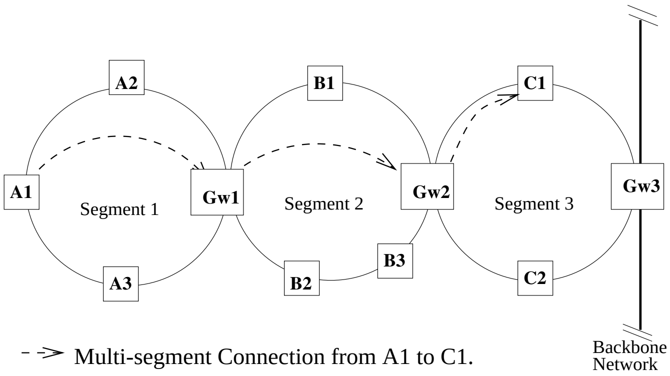

Figure 2: *A sample multi-segment* RETHER *connection from Node A1 to C1, through two intermediate switches Gw1 and Gw2.*

3.1 Connection Setup and Admission Control

In single-segment *RETHER* , connections are set up by modifying the information on the token when it arrives. It is not necessary to contact any other node, including the receiver. Multi-segment *RETHER* , however, needs an explicit connection setup protocol to establish end-to-end real-time connections across bridges/switches. *RETHER* connection messages are initiated by the senders and routed via the static routing tables in the switches. As a multi-segment *RETHER* connection is established, resources, namely buffers and network bandwidth, on the nodes along the way are reserved accordingly. More concretely, as each intermediate switch receives a connection request, it creates a single-segment *RETHER* connection on the network link through which the message is forwarded. In addition, the switch maintains the following information:

- The multi-segment *RETHER* connection ID, in the form of the sender's IP address and source port number, and the local single-segment *RETHER* connection ID.
- The next-hop interface and the Ethernet address of the next-hop node.
- The previous-hop interface and the Ethernet address of the previous-hop node.

When the last-hop switch successfully reserves a single-segment *RETHER* connection to the destination node, it sends back an acknowledgment through the same path to *commit* the resource reservations made by intermediate switches and the sender. Only when the sender receives a success acknowledgment for connection establishment will it start to send real-time data out.

In multi-segment *RETHER* , admission control is performed independently on each of the segments on a hop-by-hop basis. Each intermediate switch applies the same admission control criterion as single-segment *RETHER* . If the admission test fails in any of the segments, a connection termination message is sent on the same path back to the sender node, to cancel all resource reservations. Note that each single-segment *RETHER* connection is only aware of its other end-point in the same segment. The information about the final source and destination of the multi-segment *RETHER* connection is hidden from non-periphery switches.

3.2 Fault Tolerance

When the token on an Ethernet segment is lost or corrupted, the single-segment *RETHER* protocol's fault tolerance mechanism recovers from the fault by reintroducing the token in that segment. All the real-time connections that pass through the segment continue to work after token recovery. Therefore, multi-segment *RETHER* does not introduce any new problems compared to single-segment *RETHER* in this case. However, when network nodes crash, new mechanisms need to be devised to handle multi-segment connections in which the failed nodes participate.

For a multi-segment *RETHER* connection, either the crashed node is involved in the real-time connection or it is not. If the failed node is one of the intermediate switch or an end-point of a *RETHER* connection, the state associated with the connection needs to be cleaned up and the connection has to be reestablished, if possible. Connection re-establishment only makes sense when the failed node is one of the intermediate switches and there is an alternative route that can be used to bypass the failed switch. The cleanup of the state associated with a *RETHER* connection whose intermediate switch has crashed is triggered by the detection of this failure in all the segments to which the switch is connected. Because a *RETHER* switch participates in the token passing on all segments that are connected to it, the switch failure is detected independently on each of the segments via the fault tolerance scheme built into the single-segment *RETHER* . The nodes that detect the failure then broadcast a message to that effect on their respective segments. The other end points of the sub-connection to which the crashed node was connected, upon receiving such a message, frees up associated resources, and sends an abort message to the next sub-connection. The message eventually reaches the actual end-points of the connection in either direction and all the reserved resources for the connection are released. Just like the termination message due to failure of admission, a message travels along the path of the connection on both sides of the failed node. If the failed node is an end point of a real-time connection, the processing is similar except the clean-up message for each af-

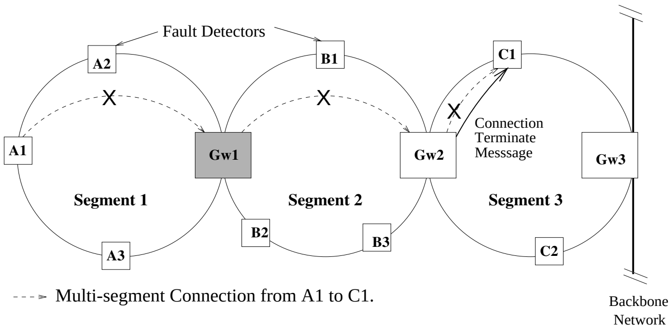

Figure 3: *Failure of an intermediate switch in a multi-segment connection. The failure is detected independently on all the segments the switch is connected to, in this case, Segment 1 and 2.*

fected real-time connection only propagates in one direction.

For instance, in Figure 3, suppose Gw1 were to crash and Node A2 detects the failure on Segment 1 and Node B2 detects it on Segment 2. A2 and B2 inform all the node on their respective segments by broadcasting a message. On receiving the message, A1 terminates its sub-connection and frees up the connection's associated resources. Gw2 similarly terminates the sub-connection whose other end-point was Gw1, on Segment 2 Since this is a multi-segment connection, Gw2 also sends a message to C1 to terminate the entire connection and to free the reserved resources. Thus, all the real-time connections that have Gw1 on their path are terminated with the associated state across the network cleaned up.

If, on the other hand, the failed node is not involved in the real-time connection, then the connection continues as before. For example, if Node A3 in Figure 3 dies, the real-time connection from A1 to C1 remains operative after the token recovery. The failure is detected and the token is recovered locally in the segment to which the failed node is connected. The effect on any real-time sessions crossing this segment is that they would not be able to send/receive data during the fault recovery period.

3.3 Buffer Management

The token cycles on adjoining network segments are not synchronized and this could lead to longer latency because of temporal skews of the token arrival times on connected segments. Since the token rotation times (TRT) in both segments are the same, the maximum skew between them could be one TRT long. For example, in the worst case, the switch receives data from the incoming segment but just missed the token on the outgoing segment. Hence, it may have to wait as much as one TRT before forwarding the data onto the outgoing segment. In the meantime, data would start arriving on the incoming segment for the next cycle, leading to buffer overflow if there is only one buffer at the switch. To avoid this situation, we use a double buffering scheme in which there are two buffers for each real-time stream going across the switch. While one buffer is being filled by the input sub-connection on one segment, the other is emptied out by the output sub-connection when the token on the outgoing segment arrives. At the end of each token cycle, the roles of the two buffers switch.

The size of the buffer has a direct effect on the end-to-end latency experienced by the applications. This implies that the buffering delays at each hop must be small and that the number of hops that a real-time connection can cross, is bounded. In theory, the token cycle times on all network segments are supposed to be the same, and in each cycle the switch receives exactly one frame of data from the incoming segment and sends out exactly one frame of data on the outgoing segment. Therefore, the real-time connection suffers a maximum of one token cycle latency at each hop along the path, and the worst-case end-to-end latency is *Number of hops * Token cycle time.* The minimum latency would be the time to transmit the data from the sender to the receiver as though they were on the same segment plus the time to copy the data into memory and out at each intermediate switch. Unfortunately, in practice, neither the network segments have identical token cycle times, nor does the switches forward the data they receives immediately.

4 Implementation and Performance

Both single-segment and multi-segment *RETHER* have been successfully implemented and tested on a prototype test-bed. The initial implementation was on FreeBSD 2.1, and has been ported to Linux and DOS. The test-bed, shown in Figure 4, consists of four Ethernet segments connected by three switches. Two of the segments are 100 Mbps Eth-

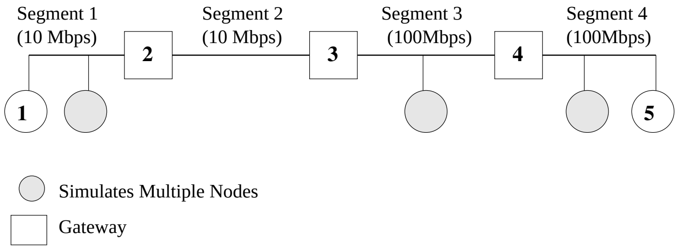

Figure 4: *The network setup for multi-segment* RETHER *experiments. There are four segments, two of them 100-Mbps and the other two 10-Mbps Ethernets.*

ernet while the other two are 10 Mbps Ethernet. The switch that connects the two 10-Mbps segments, and the non-switch machines on the 10-Mbps segment are 66MHz 486 machines, while those on the 100-Mbps segments are 90MHz and 100MHz Pentiums. Although the wiring and the NIC hardware at the hosts need not be changed, *RETHER* has to be implemented in the intermediate switches to provide bandwidth/delay guarantees. Because modifications to commercial Ethernet switches were not possible for us, we implemented the *RETHER* switch using a general-purpose machine that is equipped with multiple Ethernet interfaces, much like a network-layer software router. With the advent of faster microprocessors and system architecture, we believe implementing LAN switches based on general purpose machines is both feasible and cost-effective. Our implementation experience shows that it is indeed possible to build a *RETHER* switch completely in software. Since all the experiments are conducted locally in our lab, propagation delays are negligible in these measurements. For all the following measurements, the token cycle time is set to be 33 msec.

Extensive tests on the prototype demonstrate that bandwidth reservations made by *RETHER* connections are indeed satisfied in all cases. Since *RETHER* is implemented directly inside the device driver, each packet arrival, be it token or data, entails an interrupt processing overhead. When the network is lightly loaded and there are few nodes in the network, the token simply circulates around the network and the CPU processing overhead for token-circulation interrupts is significant. This is indicated in Figure 5. The graph plots the time taken by a user level process to execute the same computation intensive program without *RETHER* and with *RETHER* in the presence of minimal real-time bandwidth reservation. The measurements were

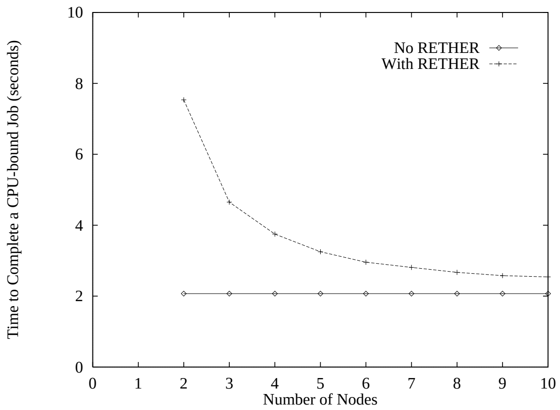

Figure 5: *As the number of nodes on the network increases, the per-node token-induced interrupt processing overhead decreases, because the token visits a node less frequently.*

made on a 100-Mbps network in which the token processing time is only 70μsec per node. As can be seen, the token-induced interrupt overhead becomes acceptable only when the 100-Mbps network has five or more nodes. On 10-Mbps networks, the relative interrupt processing overhead was not as bad because the token processing time is around 450 μsec.

No. of Hops	**No Pre-existing Connection**	**With Pre-existing Connection**
0	0.647	1.098
1	1.050	2.052
2	2.646	5.592
3	4.463	9.119

Table 1: *The time in msec to set up a real-time connection across different numbers of switches, with and without pre-existing real-time connections.*

Table 1 indicates the time to setup connections crossing 0 to 3 switches. Column 2 shows the connection setup time when all the Ethernet segments are in the CSMA mode. The main delay component in this case is the time to switch each segment from the CSMA to the *RETHER* mode. Column 3 indicates the time taken to setup a connection when the corresponding network segments are already running *RETHER* . In this case, the main component in the connection establishment time is the time to forward the connection establishment message in the non-real-time mode. The connection establishment time increases with the amount of bandwidth already reserved for real-time connections because it would take longer for the connection request message, which is transmitted as non-real-time traffic, to reach its destination. The protocol processing associated with connection setup itself at each intermediate switch is relatively minor compared to the above times. A significant component of the connection setup delay is due to scheduling and executing user processes at either end-point to complete the connection establishment. However, these are not under the control of *RETHER* and thus are not included here. The times reported in Table 1 include the time to set up the connection at the receiver and sender ends in the kernel, but do not include any user-level processing.

5 Related Work

Kopetz's MARS system [4] prototype was also focused on process control applications, but used a TDMA protocol to provide real-time guarantees on Ethernet. described a multi-token-ring protocol that is designed The token ring in Totem [5] provided ordered multicasting rather than real-time performance guarantees. Hermant [2] presented a variant of CSMA/CDR for real-time scheduling in distributed multi-access broadcast communication channels. This protocol was not meant for existing Ethernet hardware.

More recently several commercial products available in the market attempt to provide real-time performance guarantee over LANs. HP's 100VG-AnyLAN [1] uses advanced Demand Priority Access to provide users with guaranteed bandwidth and low latency, and is now the IEEE 802.12 standard for 100-Mbps networking. National Semiconductor's Isochronous Ethernet [8] includes a 10-Mbps P channel for normal Ethernet traffic, 96 64-Kbps B channels for real-time traffic, one 64-Kbps D channel for signaling, and one 96-Kbps M channel for maintenance. The 96 B channels can provide bandwidth guarantee to network applications because they are completely isolated from the CSMA/CD traffic. Isochronous Ethernet forms the IEEE 802.9 standard. 3COM's Priority Access Control Enabled (PACE) [3] technology enhances multimedia (data, voice and video) applications by improving network bandwidth utilization, reducing latency, controlling jitter, and supporting multiple traffic priority levels. PACE technology uses star-wired switching configurations and enhancements to Ethernet that ensure efficient bandwidth utilization and bounded latency and jitter. Because the real-time priority mechanism is provided by the switch, there is no need to change the network hardware on the desktop machines. More recently, there are 802.1p and 802.1q efforts that support packet prioritization and virtual LANs. Peterson [6] summarized the current efforts in the Industrial Automation community to use Ethernet as the control network technology.

The difference between *RETHER* and all the above work is that *RETHER provides bandwidth/delay guarantees* to network packets, rather than just supports packet prioritization. In addition, all of the other schemes require changes to the existing infrastructure in the host network hardware and/or the wiring, while *RETHER* does not. *RETHER* 's ability to use commodity Ethernet hardware is crucial for industrial automation systems to ride with the technology momentum of the PC networking industry. Finally, *RETHER* is the only system that provides bandwidth guarantees for real-time connections that run cross multiple hops, an critical feature for system scalability.

6 Conclusion

This paper presents the design, implementation, and evaluation of single-segment and multi-segment *RETHER* protocols. The major contribution of this work is a simple and efficient traffic control mechanism that turns commodity off-the-shelf Ethernet hardware into a network capable of providing delay/bandwidth guarantees, and thus makes it possible to deploy Ethernet to PLC networks used in industrial automation systems. Because *RETHER* is a software-only solution, no hardware modification is required. *RETHER* is also innovative because it supports both real-time and non-real-time traffic in a single framework, and because it is scalable to a large number of PLC nodes with its built-in end-to-end performance guarantee mechanism.

RETHER has also been extended to wireless LAN with roaming support [7]. In addition, we have also developed a new real-time Ethernet switch architecture called *EtheReal* [9], which takes one step further by providing bandwidth/delay guarantees without requiring any changes to both software and hardware on the host ends. We are currently developing a middleware based on the distributed shared memory abstraction and *RETHER* , with the goal to simplify the development of distributed real-time embedded systems. More up-to-date information about the *RETHER* project can be found in `http://www.ecsl.cs.sunysb.edu/rether.html`.

Acknowledgments

This research is supported by an NSF Career Award MIP-9502067, NSF MIP-9710622, NSF IRI-9711635, NSF EIA-9818342, NSF ANIR-9814934, a contract 95F138600000 from Community Management Staff's Massive Digital Data System Program, USENIX student research grants, as well as fundings from Sandia National Laboratory, Reuters Information Technology Inc., and Computer Associates/Cheyenne Inc.

References

[1] A.R. Albrecht and P.A. Thaler. Introduction to 100VG-AnyLAN and the IEEE 802.12 local area network standard. *Hewlett-Packard Journal*, 46(4), Aug. 1995.

[2] J.-F. Hermant, G. Le Lann, and N. Rivierre. A general approach to real-time message scheduling over distributed broadcast channels. *roceedings 1995 INRIA/IEEE Symposium on Emerging Technologies and Factory Automation. ETFA'95*, pages 191–204, Oct. 1995.

[3] The 3COM Technical Journal. 3Com's New PACE Technology. *http://www.3com.com/files/mktg/pubs/3tech/195pace.html*, 1996.

[4] H. Kopetz, A. Damm, C. Koza, M. Mulazzani, W. Schwabl, C. Senft, and R. Zainlinger. Distributed fault-tolerant real-time systems: the mars approach. *IEEE Micro*, 9(1):25–40, Feb. 1989.

[5] L.E. Moser, P.M. Melliar-Smith, D.A. Agarwal, R.K. Budhia, C.A. Lingley-Papadopoulos, and T.P. Archambault. The totem system. *Digest of Papers, Twenty-Fifth International Symposium on Fault-Tolerant Computing*, pages 61–66, Jun. 1995.

[6] C. Peterson. Open networking fuels the next major advancement in industrial automation. *EE Times*, Sep. 14 1998.

[7] P. Pradhan and T. Chiueh. Real-time performance guarantees over wired and wireless lans. *Proceedings of the 4th IEEE Real-Time Technology and Applications Symposium*, June 1998.

[8] Xiaonong Ran and W.R. Friedrich. Isochronous LAN based full-motion video and image server-client system with constant distortion adaptive DCT coding. *Proceedings of the SPIE - The International Society for Optical*, 2094:1030:41, 1993.

[9] S. Varadarajan and T. Chiueh. Ethereal: A host-transparent real-time fast ethernet switch. *Proceeding of International Conference on Network Protocols*, October 1998.

[10] Chitra Venkatramani. *The Design, Implementation and Evaluation of RETHER: A Real-Time Ethernet Protocol.* PhD thesis, State University of New York at Stony Brook, December, 1996.