Design and Implementation of a Real-Time Switch for Segmented Ethernets

Chitra Venkatramani
T.J.Watson Research Center
IBM
Hawthorne, NY 10532

Tzi-cker Chiueh
Dept. of Computer Science
SUNY at Stony Brook
Stony Brook, NY 11794

Abstract

Providing network bandwidth guarantees over an Ethernet requires coordination of the network nodes for traffic prioritization such that real-time data can have deterministic access to the network. We have shown previously how RETHER, a software-based token passing protocol can efficiently provide real-time support over a single shared Ethernet segment. This work extends the token passing mechanism into a switched, multi-segment Ethernet environment. This paper describes the detailed design issues, their solutions, and a fully operational switch implementation built into the FreeBSD kernel. By testing the protocol independently and as the underlying network protocol of the Stony Brook Video Server, we have verified that the bandwidth guarantees are successfully provided, with relatively low run-time overhead, for real-time connections crossing multiple Ethernet segments. This paper also provides a comprehensive performance evaluation of the prototype.

1 Introduction

Distributed multimedia applications demand time-constrained data delivery, which requires resource guarantees from underlying networking and I/O systems. In previous papers [6] and [14], we described RETHER, a software-only real-time protocol that provides network bandwidth guarantees to multimedia applications on a single shared Ethernet. Although Ethernet performs well for non-real-time traffic, providing bandwidth guarantees on an Ethernet is problematic for two reasons. The first reason is that Ethernet [1] uses a nondeterministic exponential backoff algorithm to resolve collisions on the channel while attempting to transmit packets. This nondeterminism is unsuitable for real-time applications. The second reason is that Ethernet does not prioritize packets. This is undesirable for real-time systems in which important packets should not be held up waiting for unimportant packets. RETHER has been designed to address the above problems and to provide deterministic and periodic access to the network for multimedia applications. Because of the enormous popularity of Ethernet, it is essential to be able to support real-time services using existing Ethernet infrastructure. Hence, the RETHER protocol is built into the network device driver in the operating system, thereby operating seamlessly with any off-the-shelf Ethernet hardware.

RETHER uses a deadline-driven token passing protocol that prioritizes real-time over non-real-time data and provides deterministic channel access to real-time data. RETHER also features a distributed admission control policy to allow dynamic addition and removal of real-time connections. The fault tolerance mechanism protects the network against token loss due to node failures or bit corruptions. RETHER has been implemented and tested independently, as well as in the context of the Stony Brook Video Server [7].

The main motivation for the multi-segment RETHER work presented in this paper is the prevailing popularity of Ethernet switches, which, together with 100-Mbps Fast Ethernet and possibly Gigabit Ethernet technology, have driven multi-segment Ethernets to be deployed widely in LANs. Although quality-of-service guarantees are not possible with conventional Ethernets, our research results show that RETHER is capable of providing end-to-end bandwidth guarantees over LANs based on existing Ethernet hardware.

It is important to distinguish the protocol layer at which multi-segment RETHER works from RSVP [17], which is an emerging Internet standard that is designed to make real-time performance reservations over wide-area networks. Figure 1 illustrates where RETHER fits in the end-to-end bandwidth reservation framework. In an end-to-end connection across

0-8186-8061-X/97 $10.00

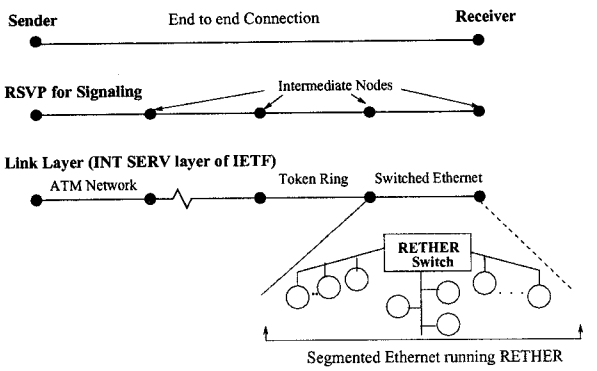

Figure 1: *A global view of an internet bandwidth reservation that uses RETHER for bandwidth reservation over the Ethernet segment.*

the Internet, RSVP is used for signaling. RSVP in turn relies on *link-layer* reservation mechanisms to actually setup the underlying connection. One of the subnetworks that participates in the real-time connection in Figure 1 is a switched multi-segment Ethernet while the others are ATM, FDDI, etc. The abstraction that multi-segment RETHER exposes is a real-time capable Ethernet subnetwork that can provide similar bandwidth guarantees. Hence, RETHER plays a complementary role to higher level reservation protocols in that it functions exclusively as a link-layer building block.

The multi-segment implementation does not require any changes to the wiring and the hardware at the hosts. But, intermediate switches must implement RETHER for bandwidth 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 software router.

The rest of the paper is organized as follows. Section 2 reviews previous efforts for network performance guarantees at the data link and transport layers. Section 3 reviews the important features of the single-segment implementation of RETHER to set the stage for subsequent discussions. Section 4 describes the detailed design of the multi-segment RETHER protocol. Our implementation experiences and the changes we introduced to the design to address some of the problems encountered are presented in Section 5 and a detailed performance evaluation of the prototype implementation is presented in Section 6. Finally, Section 7 concludes the paper with a summary of the major research results and a discussion of ongoing work.

2 Related Work

Although there have been various proposals related to bandwidth guarantees using the token passing mechanism, we will only survey those proposals that are actually implemented. Several commercial products available in the market attempt to provide real-time performance guarantee over LANs. HP's 100VG-AnyLAN [4] 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 [12] which forms the IEEE 802.9 standard, isolates the Ethernet non-real-time data traffic from the real-time traffic by using separate channels. 3COM's Priority Access Control Enabled (PACE) [11] technology enhances multimedia (data, voice and video) applications by 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 are no changes to the network hardware on the desktop machines. Like PACE, RETHER also provides packet priorities, but, by using a token passing scheme, it eliminates nondeterminism due to collisions and backoff. The IEEE 802.4 Token Bus protocol [2] and the IEEE 802.5 Token Ring protocol [3] have mechanisms built in to permit traffic priorities. As a software solution over Ethernet, it was not feasible to adapt these protocols because of their complexity. The mechanisms used in RETHER are designed for Ethernet and are simpler because in the event that RETHER cannot recover from some fault scenarios, the hosts can always switch back to communicating using the CSMA/CD protocol that the hardware implements. RETHER provides a guarantee on network bandwidth with the support of packet priorities. All of the other schemes require changes to the existing infrastructure in the host network hardware and/or the wiring, while RETHER does not. With the multi-segment extension, RETHER provides bandwidth guarantees for real-time connections that cross multiple switches or hops.

Other mechanisms for providing priorities in CSMA/CD protocol are described in [8, 16] but require hardware support. Implementation efforts for bandwidth reservation over wide-area networks include the Tenet Protocol Suite [5], resource ReSerVation Protocol (RSVP) [17], and Stream Protocol (ST-II) [9], [10], [13]. RETHER borrows some of the ideas from these protocols in connection setup and fault tol-

erance. However, these protocols are transport-layer protocols and assume that the underlying datalink layer can provide some kind of bandwidth guarantee or latency bounds. RETHER forms the bandwidth reservation building block on Ethernet for higher layer reservation protocols and is thus complementary to these efforts.

3 Single-Segment RETHER Overview

This section briefly describes the features of single-segment RETHER that are essential to understand the multi-segment extension. The RETHER protocol 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. Under RETHER, the network operates using the original Ethernet protocol (CSMA/CD) until an application process on some network node makes the first real-time reservation request. Upon this request, the network switches to a token-passing mode. It is necessary that all nodes in the network run the RETHER kernel and switch to the token-passing protocol. In this mode, channel access for all traffic, real-time and non-real-time, is regulated by a token. Time is divided into cycles and in each cycle, traffic is prioritized. The token visits all nodes that have real-time data to send and then it attempts to visit *all* the network nodes in a round-robin fashion to allow them to send non-real-time data. Non-real-time nodes get to use the bandwidth remaining in each cycle after all the real-time nodes have been serviced. The network stays in the token-passing mode until the last real-time session terminates. At this time, the network switches back to the CSMA/CD protocol. This hybrid scheme reduces the performance degradation of non-real-time traffic when no real-time connection is active, and eliminates the delays due to token passing for higher level applications using the network.

The token passing works with the token beginning each cycle with a counter indicating the token cycle time. As the token is passed around, each node subtracts the time it used up, from this counter. When the counter goes down to zero, the token begins a new cycle starting with the first real-time node. RETHER implements a fault detection and recovery mechanism to protect the network from token losses. This works with node monitoring its successor in the token schedule using an acknowledgement timeout mechanism. If a node does not hear from its successor within this interval, it times out and generates a new token. RETHER implements a distributed admission control

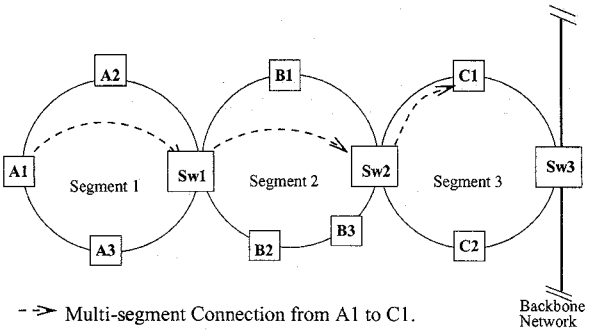

Figure 2: *A multi-segment RETHER connection from Node A1 to C1, through switches Sw1 and Sw2.*

algorithm which admits real-time sessions within the available bandwidth. This algorithm also sets aside a fraction of the total bandwidth to prevent non-real-time traffic from starvation. More details about single-segment RETHER can be found in [14].

4 Multi-Segment RETHER

To provide end-to-end network bandwidth guarantees between any pair of nodes in a multi-segment Ethernet, RETHER has been extended to operate across switches. Conceptually, the system applies the single-segment RETHER protocol on each of the segments on the path from the source to the destination of a real-time connection. The per-segment reservations are all part of one logical connection used for resource allocation. All of the Ethernet segments run the single-segment RETHER protocol , with the token cycles on different segments being independent of each other. That is, there is no synchronization requirement on the token cycles associated with adjoining segments. The following sections describe the functionality of each of the major modules of the multi-segment RETHER protocol in detail.

4.1 Connection Setup and Admission

Multi-segment RETHER includes a link layer connection setup protocol to manage connections across switches. RETHER uses the available route-discovery mechanism in the switches to discover the other end-point in the Ethernet subnetwork. This link-layer connection establishment is independent of the network-layer real-time connection establishment across the subnetwork.

Connection establishment in RETHER is receiver-initiated. The application or the network-layer reser-

vation protocol such as RSVP is responsible to determine the sender's address and bandwidth characteristics. This information is then conveyed to RETHER to enable it to establish the link-layer connection. The connection request is sent as a special message that is intercepted by RETHER modules along the path from the receiver to the sender. This is done to make resource reservations, namely network bandwidth and buffers along the way. At each intermediate switch, the next hop network and the Ethernet address of the next hop destination are saved as part of the connection state. This fixes the route that the data has to take once the end-to-end connection has been admitted.

If the connection setup should fail for reasons such as insufficient buffer memory, failure to admit the connection, or no route to the destination, then a non-real-time failure message retraces the path of the original message to release all previously allocated resources. An error is returned to the application which may attempt to establish the connection again.

In Figure 2, the connection request is generated by C1 and travels to A1 through Sw2 and Sw1 establishing subconnections Sw2-C1, Sw1-Sw2 and A1-Sw1 in segments 3, 2 and 1 respectively. Admission control is performed independently on each of the segments using the same algorithm as in single-segment RETHER. It is based on guaranteed service for the bandwidth requested by the application. This module is hence a straightforward extension of the one in the single-segment RETHER protocol. Once the connection is admitted in the last Ethernet segment, it is complete and data transfer begins immediately. There is no explicit *connection accept* message when the connection is admitted. Instead, the fact that the receiver receives the first frame of data acts as an implicit acknowledgment.

4.2 Fault Tolerance

When the token on an Ethernet segment is lost due to bit-errors, 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 poses no new problems compared to single-segment RETHER in this case. However, when nodes crash, new mechanisms need to be devised to handle multi-segment connections in which the failed nodes are involved.

For a multi-segment connection, if the crashed node is part of an RT connection, the state associated with

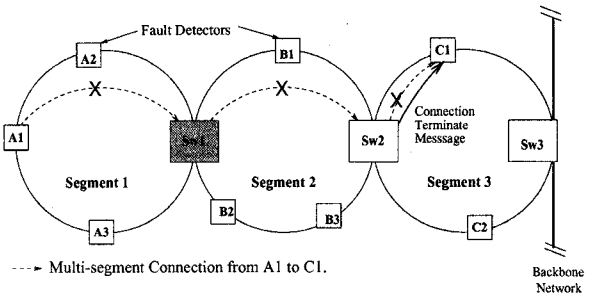

Figure 3: *Failure of an intermediate switch in a multi-segment connection. The failure is detected independently in segments 1 and 2.*

the connection needs to be cleaned up across the network. When an intermediate switch crashes, it is detected independently in each of its connected segments by the fault detection mechanism of single-segment RETHER described in 3. 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. For an intermediate switch crash, a termination message travels along the path of the connection on both sides of the failed node whereas for the crash of an end-point, the message travels only in one direction. Figure 3 illustrates this mechanism when Sw1 crashes and A2 and B1 detect the failure.

If the failed node is not involved in a real-time connection, then the connection continues as before. The failure is detected and the token is recovered locally in the segment to which the failed node is connected. The only effect on connections passing through that segment is that the senders would not be able to send any data during the token recovery period.

4.3 Buffer Management

The token cycles on adjoining network segments are not synchronized and this could lead to longer latency because of temporal skew of token arrival times. Since the token rotation times (TRT) in both segments are the same, the maximum skew between them could be one TRT long. To handle this skew, 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 filled by one sub-connection on one seg-

ment, the other is emptied out by the sub-connection on the next segment. At the end of each token cycle, the roles of the two buffers are swapped.

The size of the buffer has a direct effect on the end-to-end latency seen by the users. The buffering delays create interactivity problems for real-time teleconferencing applications where maximum one-way delays can be in the range of a few hundred milliseconds. This implies that 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 maximum 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 to memory and out at each intermediate switch. However, in practice, neither the network segments have identical token cycle times, nor does the switch forward the data it receives immediately. The delay impact of the discrepancies in token cycle times and the possible approaches to solve the problem are described in Section 5.

Although theoretically two buffers are sufficient at the switches to take care of the token cycle skew, in practice, buffers are also required to take care of fault recoveries (See [15] for details). In our implementation the maximum number of frames that can accumulate at a switch for a particular multi-segment stream is determined by the end-to-end latency specified for the stream.

5 Implementation Experiences

After implementing the multi-segment RETHER protocol as described in the previous section, several unexpected problems surfaced, which forced us to go back to devise additional mechanisms to supplement the original design. This section describes these modifications and additions.

The main problem that we encountered in our first implementation was a significant deviation of the token cycle times from the ideal values, on adjoining segments. This led to serious problems of buffer overflow/underflow at the switches, and a breakdown of bandwidth guarantees for multi-segment and single-segment connections. We traced the deviation and found that it was due to the inaccuracy in determining the token processing time at the switch. There was contention for the CPU due to packets arriving on multiple segments more or less simultaneously. Because of the contention, the processing of tokens/messages on one segment is delayed by the switch's processing of the token or message on another segment. Hence, the token holding time and processing times were larger than the predetermined values used in our initial implementation. This led to differences between the average token rotation times (TRT) of the adjoining segments. For a connection that crosses a pair of such segments, this phenomenon allows the segment with the smaller average TRT to send data at a faster rate than the segment with the larger TRT can consume, leading to buffer overrun problems at the switch. Because of this problem, the source sends data at the normal rate but data is delivered to the client only at the rate of the slowest link in the path of the connection, which results in the loss of data and violation of end-to-end bandwidth guarantees.

The two methods we considered to accommodate the nondeterminism are – i) remove it by carefully scheduling token and data arrivals at the switch or ii) to keep the token cycles on adjoining segments independent and devise a compensation algorithm that dynamically measures the nondeterminism and accounts for it.

We chose the latter alternative because keeping the token cycles on adjoining segments independent of one another is advantageous in many ways. This scheme decouples the operations of segments and has the very desirable effect of *fault isolation*. That is, the effect of a fault can be confined to the originating segment and will not propagate to other segments, and the error recovery process for each segment can be performed completely locally. Secondly, the token cycles on the two segments can be skewed from each other and need not be synchronized, as would be required if scheduling were performed at the switch. Thirdly, the performance overhead of recomputing schedules on addition and termination of streams is completely eliminated. Finally, the complexity and inefficiency problem of a scheduling algorithm grows as the number of network links attached to the switch increases. Because the current implementation of RETHER switch is on a general-purpose machine, the no-scheduling policy significantly improves the scalability of RETHER. It is also worth noting that the no-scheduling policy still

ensures a maximum packet latency of one token cycle at each intermediate switch.

5.1 Compensation Algorithm

We developed a history-based prediction mechanism to estimate the discrepancy between the ideal token cycle time and the actual token cycle time in the next cycle. One of the nodes in each segment is designated the *synchronizer* node and is supposed to receive the token at the beginning of each token cycle. The *synchronizer* node takes a time stamp at each token visit and if the token cycle time is not the ideal token cycle time, then it applies a correction to the token cycle time in the next cycle to take care of the detected discrepancy. The correction term is computed based on the correction applied in the previous n cycles[1].

In the current design, all the RT nodes independently compute the correction term. However, only the first real-time node in the token cycle serves the *synchronizer* role and actually applies the correction. Because the list of real-time nodes is stored in the token, it is straightforward for each node to decide whether it is the *synchronizer* node. In other words, there is no need to re-elect a new synchronizer node when the previous synchronizer dies or all RT streams from that previous synchronizer terminate. The cycle-time prediction and adjustment algorithm executed by all RT nodes is shown in Figure 4.

Token_Residual_Time indicates the residual time counter on the token that indicates the time remaining in the current token cycle. *Difference* is the token cycle discrepancy calculated from the previous cycle. *Compensation* is the cumulative errors observed so far. The initial *Token_Residual_Time* for the next cycle is computed by subtracting *(K * Compensation)* from *Ideal_Token_Cycle (33msec)*, where K is the damping factor to avoid over-compensation. A larger value of K implies that any deviation observed in the cycle is compensated for quicker. A smaller value for K indicates that the compensation is done slower. We examine how the choices of K affects the token cycle time accuracy in Section 6. The *gettime()* in line 1 of the algorithm determines the actual time when the token arrives at this node and is used to compute the duration of the previous cycle. Intuitively, this time stamp should also mark the beginning of the next token cycle. However, if the token happens to come early and a *Delay* is done to compensate for the early arrival, the actual start of the next cycle is after the *Delay*. Hence, we take another time stamp at the end of the processing, on line 12. This algorithm has been implemented in our multi-segment prototype and as a result the cycle times across various segments remain stable with very small jitter. This in turn ensures smooth delivery of data for multi-segment real-time connections, as demonstrated in Section 6.4.

[1] The value of n determines the amount of history used in prediction and is a configurable parameter.

6 Performance Evaluation

The goal of implementing RETHER is to show how one can support real-time guarantees in an Ethernet switch at all and since it is a software implementation the performance of the RETHER switch prototype is not comparable to that of a commercial hardware switch. We believe that implementing a RETHER switch in hardware will provide comparable performance.

The set of measurements for multi-segment RETHER can be divided into two categories. The first measures the performance of an RT connection including the connection setup time and the end-to-end latency. The second evaluates the effectiveness of a general purpose machine acting as a switch to send/receive data on multiple network interfaces, such as the effective cross-section bandwidth that such a switch can deliver.

6.1 Experimental Setup

The setup of networks for our experiments is shown in Figure 5. There are four Ethernet segments connected by three switches. Two of the segments have a bandwidth of 100-Mbps and the other two have a bandwidth of 10-Mbps. 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. Since all the experiments were conducted locally in our lab, propagation delays are negligible in these measurements. For all measurements, the token cycle time is set to be 33 msec.

Since RETHER is an implementation inside the operating system, each packet arrival, be it token or data, entails an interrupt processing overhead. As described in [15], the interrupt overhead due to token passing in a lightly loaded network is very significant in small networks. In a 100Mbps network, where the idle token processing overhead is 70μsec, the interrupt overhead became tolerable compared to the case when RETHER was not running, only when the network

```
1       Current_TimeStamp = gettime();
2       if (NodeID == Synchronizer) {
3               Difference = Current_TimeStamp + Token_Residual_Time
                                - Prev_TimeStamp - Ideal_Token_Cycle;
4               if (Difference < 0) {
5                       Delay a time period of duration abs(Difference);
6               }
7               /* accumulate errors from previous runs */
8               Compensation += Difference;
9               if (Compensation < 0)
10                      Compensation = 0;
11              Token_Residual_Time = Ideal_Token_Cycle - K * Compensation;
12              Current_TimeStamp = gettime();
13      }
14      Prev_TimeStamp = Current_TimeStamp;
```

Figure 4: *Cycle time compensation algorithm.*

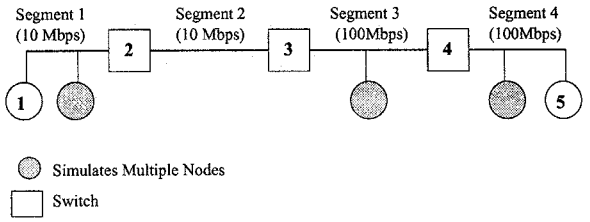

Figure 5: *The setup for multi-segment experiments.*

had five or more nodes. Hence, for our experiments we consider networks with five or more nodes. Due to lack of enough PC's to populate four network segments for the multi-segment experiments, we created each network with at least three nodes including the switch. The third node just simulated multiple network nodes by holding the token for multiple token times during each token visit and none of the measurements were done on such nodes. Figure 5 shows such nodes as shaded circles.

6.2 Connection Setup Time

The first set of experiments measure the delay of setting up an end-to-end real-time connection across switches. Table 1 indicates the time to setup connections crossing 0 to 3 switches. Column 2 shows the time taken when the first real-time connection across the various hops is established, i.e., when RETHER is not running. The main component in these times is the time to make each of the segments involved switch 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 real-time reservation made because it would take longer for the connection request message to reach its destination in the non-real-time mode. The protocol processing associated with connection setup itself, at each switch is relatively small compared to the above times. All measurements were in the kernel and the delay associated with the user process at the two end-points are not included in these measurements. Since the switches in the experiments were heterogeneous, it took different amounts of time to perform the connection setup on different switches.

No. of Switches	RETHER not running (msec)	RETHER running (msec)
0	0.647	1.098
1	1.050	2.052
2	2.646	5.592
3	4.463	9.119

Table 1: *The connection setup time across different number of switches.*

6.3 End-to-end Latency

The multi-segment RETHER protocol avoids scheduling of token cycles on adjacent segments. Instead, it relies on buffering at the intermediate switches to decouple the token cycles on adjoining network segments. Hence, it is important to examine the impact of switch buffering on the end-to-end packet latency. Since we use a double buffering scheme, when the overflow buffers are empty, the maximum buffering delay at each intermediate switch is one token cycle time. The maximum end-to-end packet latency is therefore the product of the number of hops and the

token cycle time. The actual buffering delay at the switch depends on the relative timing between the RT token visits on the two subnets. It is essentially the timing difference between when a particular frame is received on one subnet and when it is forwarded on the other. It could therefore be less than one token cycle time. For many sample runs, the end-to-end latency was determined to be well within the maximum latency bound of one TRT per switch. The maximum and typical measured values for one sample connection are indicated in Table 2.

No. of Switches	Measured (msec)	Maximum (msec)
1	19.18	33
2	39.77	66
3	63.65	99

Table 2: *The measured and maximum values for end-to-end packet latency as the number of intermediate switches increases.*

6.4 Cycle Time

The next performance metric is to determine if the bandwidth guarantees are indeed met for multi-segment real-time connections. We measure the mean token cycle time in each segment and also the standard deviation from the ideal cycle time of 33 msec, for a real-time connection from Node 1 to Node 5.

Our initial implementation did not have the token cycle time compensation algorithm described in Section 5.1. On conducting experiments without the cycle time compensation turned on, we observed that there was a wide variation in the cycle times of the different segments. This is indicated in Row 1 of Table 6.4, corresponding to K=0. The variation in cycle times is predominantly due to the inaccuracy in estimating token holding time at the switch.

The effect of this variation in cycle times on the end-to-end connection was in the form of – i) lost frames due to buffer overflow and ii) jitter due to buffer underflow at the switches. In this particular run, with K=0, 3000 frames were sent from the sender to the receiver. Of these, 409 frames were lost due to buffer overflow and the jitter ranged between 33.67 and 135.54 msec.

However, when the compensation mechanism is built in, the average cycle times of the different segments are very close and the standard deviation values are much smaller. As shown in Table 6.4 we also measured the cycles times for different values for the damping factor K. The goal is to determine empirically the optimal choice of K in order to maintain a low standard deviation of the cycle time. A smaller deviation ensures that the data flow across the switches is smooth and all the real-time data reaches the destination node with a predictable timing. Table 6.4 shows smaller values of K are favorable.

The effect of K can be understood by examining Figure 4. A larger value of K indicates a more aggressive compensation strategy whereby, any variation in cycle time is assumed to be long-lived and is compensated for quicker. Smaller values of K indicate a more conservative compensation of the token cycle time. When a multi-segment stream starts up for the first time, it takes a few cycles to measure the error and for compensation to take effect and for the actual token cycle time to settle. However, once this phase is over, the effect of the value of K can be seen only when there are instantaneous phenomena causing variations in the cycle time. The stability of the cycle time for small values of K indicates that any variations during the period that there are real-time streams are very short-lived and an aggressive cycle time compensation is not called for. This is validated by the higher standard deviation values when $K = 1.0$.

The measurements indicate that the multi-segment RETHER protocol indeed works as expected and the accuracy of the token cycle time is not affected by the number of switches that the real-time connection has to cross.

6.5 Cross-Section Bandwidth

An important criterion for the evaluation of a switch is its cross-section bandwidth. Ideally, a switch should have the capability of passing bits from one segment to the other, without any performance degradation. Since the RETHER switch is a general purpose machine performing all switch functionality in software, the performance cannot be as good as the ideal situation. This is mainly because all token arrivals generate interrupts and interrupt processing overheads are significant. Moreover, there is also resource contention for the I/O bus and memory when data packets come over multiple networks at a switch as compared to a non-switch machine. Our measurements indicated that the number of MPEG-1 streams that could be supported across a Pentium 100 switch was 39 MPEG-1 streams or 86% of the single-segment maximum of 45 streams. This corresponds to 59 Mbps. For a 66MHz i486 machine connected by two 10-Mbps Ethernet links, the cross-section bandwidth is measured to be 6.45 Mbps, which is 73% of the single-segment measured maximum of 8.8 Mbps.. The

Damping Factor K	Segment 1		Segment 2		Segment 3		Segment 4	
	Mean	Dev.	Mean	Dev.	Mean	Dev.	Mean	Dev.
0.0	38.07	4.75	41.46	8.22	35.17	1.92	33.88	0.56
0.2	33.49	0.59	33.38	0.57	33.52	0.38	33.35	0.08
0.5	33.49	0.56	33.41	0.52	33.52	0.39	33.35	0.08
0.8	33.59	0.75	33.49	0.65	33.55	0.44	33.35	0.09
1.0	33.73	1.05	33.61	0.85	33.59	0.49	33.36	0.09

Table 3: *Variation of the mean cycle time and the standard deviation in msec, with the damping factor K.*

setup for this experiment had one host on either side of the switch, with one acting as the receiver for all the streams and the other acting as the sender of all the streams.

We conducted experiments to determine the cause for the reduced cross-section bandwidth and determined that the Pentium 100 CPU was the bottleneck. This was mainly due to the excessive interrupt processing overhead at the switch. When we later replaced the switch with a Pentium 200, we found that the switch was no longer the bottleneck.

6.6 Fault Recovery Time

The multi-segment RETHER protocol provides a fault recovery mechanism that cleans up the connection state and reclaims all the reserved resources for a connection when any node that is part of the connection crashes. The node that crashes could be an intermediate switch or a connection end-point.

The failure of a switch is detected independently in both the segments that it is connected to and a connection termination message travels along the path of connection, in both directions. This message causes each of the intermediate nodes in the multi-segment connection to free all the reservations and clean up all the state associated with the real-time connection.

The experiments to verify this were performed by crashing each of the five nodes numbered 1 to 5 in Figure 5, for one real-time connection from Node 1 to Node 5. Row 1 in Table 4 indicates the recovery time when the sender end point was crashed. Rows 2, 3 and 4 are when the three intermediate switches were crashed and finally, Row 5 indicates the recovery time when the receiver end-point of the connection is crashed.

Each entry in Table 4 indicates the recovery time at a particular node once it receives a connection termination message. Each of the entries in Table 4 consists of one or two values separated by a '+'. The first is always the time to recover from the fault while the second indicates the time to forward the connection termination message. The total time to recover an end-to-end connection when one of its intermediate switches fails is the maximum of time to recover the connection on either side of the crashed node. This is because fault recovery proceeds in parallel along both directions. In Table 4, the time to recover from a fault is therefore the maximum of the sum of the values to the right and the left of the X.

Table 4 clearly shows how the connection termination message travels along both directions starting from the crashed node. In general, for each of the real-time connections that pass through the crashed switch, RETHER's fault recovery mechanism recovers the reserved resources across all the segments involved in the connection.

7 Conclusion

We show in this paper that it is possible to efficiently support end-to-end bandwidth guarantees over a local inter-network composed of Ethernet segments connected by switches. With switched Ethernet, Fast Ethernet and Gigabit Ethernet entering the mainstream, it seems like Ethernet is gaining popularity over ATM in the workgroup networking area. In this context, we expect the experiences and results from the RETHER project to play a crucial role in moving the Ethernet-based real-time technologies forward. This paper presents the design, implementation, and evaluation of a multi-segment RETHER protocol, which provides an end-to-end bandwidth guarantee service to user applications across multiple Ethernet segments. Through a detailed evaluation of the prototype, we also show that the implementation is reasonably efficient in a testbed that consists of 10-Mbps and 100-Mbps links, and the target performance guarantees are indeed met in all cases.

There are several directions that the RETHER group is currently pursuing actively. First, we would like to experiment the switch implementation on a high-end Pentium machine that has more efficient I/O hardware to evaluate the performance scalability of the switch, in terms of larger numbers of attached links as well as higher link bandwidth. In addition,

Failed Node	Recovery Time (msec)				
	Node 1	Node 2	Node 3	Node 4	Node 5
1	X	45.71+25.03	17.73	15.37	0.17
2	41.37+2.01	X	47.94+19.31	16.69	0.16
3	15.95	46.43+1.82	X	49.72+33.23	0.16
4	11.79	16.92	46.40 + 1.58	X	46.60 + 1.82
5	12.86	21.63	15.43	44.94+2.31	X

Table 4: *Time to process a fault recovery message at nodes/switches when a fault occurs. X indicates failed node.*

the software architecture will use polling instead of being interrupt-driven, to minimize the CPU overhead. Since wireless LANs have a physical network architecture similar to Ethernet, we are planning to extend the multi-segment RETHER technology to a heterogeneous network of wired and wireless links. Finally, we are looking at the feasibility of implementing part of the multi-segment RETHER protocol in hardware.

References

[1] Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications. *ISO/IEC 8802-3, (4th edition)*, (1993-07-08).

[2] IEEE/ANSI Standard 802.4 - 1985, Token passing bus access method and physical layer specifications. *IEEE Inc., New York*, 1985.

[3] IEEE/ANSI Standard 802.5, Token ring access method and physical layer specifications. *IEEE Inc., New York*, 1985.

[4] 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.

[5] Anindo Banerjea et al. The Tenet Real-Time Protocol Suite: Design, Implementation and Experiences. *IEEE/ACM Transactions on Networking*, 4(1):1:10, Feb. 1996.

[6] T. Chiueh and C. Venkatramani. Supporting Real-time Traffic on Ethernet. *Proceedings of IEEE Real-time Systems Symposium*, Dec. 1994.

[7] T. Chiueh, C. Venkatramani, and M. Vernick. Design of the Stony Brook Video Server. In *SPIE First International Symposium on Technoloigies and Systems for Voice, Video and Data Communications*, Philadelphia, PA, Oct 1995.

[8] R. Court. Real-time Ethernet. *Computer Communications*, pages 198–201, Apr. 1992.

[9] L. Delgrossi, R.G. Herrtwich, and F.O. Hoffmann. An implementation of ST-II for the Heidelberg Transport System. *Internetworking: Research and Experience*, 5(2):43:69, June 1994.

[10] R. Herrtwich and L. Delgrossi. Beyond ST-II: Fulfilling the requirements of multimedia communication. *Network and Operating System support for digital audio and video*, Nov. 1992.

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

[12] 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.

[13] C. Topolcic. Experimental internet stream protocol, Version 2 (ST-II). *Internet RFC:1190*, Oct. 1990.

[14] C. Venkatramani and T. Chiueh. Design, implementation and evaluation of a software-based real-time Ethernet protocol. *ACM SIGCOMM 95*, 1995.

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

[16] R. Yavatkar. A reservation-based CSMA protocol for integrated manufacturing networks. *IEEE Transactions on Systems, Man and Cybernetics*, 24(8), Aug. 1994.

[17] Lixia Zhang, Steve Deering, Deborah Estrin, Scott Shenker, and Daniel Zappala. RSVP : A New Resource Reservation Protocol. *IEEE Network Magazine*, Sept. 1993.