Achieving Real-Time Communication over Ethernet with Adaptive Traffic Smoothing

Seok-Kyu Kweon and Kang G. Shin
Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109-2122
{skkweon,kgshin}@eecs.umich.edu

Gary Workman
General Motors Tech Center, Warren, MI 48090-9040
workmangc@aol.com

ABSTRACT

Ethernet continues to be one of the most popular LAN technologies. Due to the low price and robustness resulting from its wide acceptance and deployment, there has been an attempt to build Ethernet-based real-time control networks for manufacturing automation. However, it is difficult to build a real-time control network using the standard UDP or TCP/IP and Ethernet, because the Ethernet MAC protocol, the 1-persistent CSMA/CD protocol, has unpredictable delay characteristics. When both real-time (RT) and non-real-time packets are transported over an ordinary Ethernet LAN, RT packets from a node may experience a large delay due to (i) contention with non-RT packets in the originating node and (ii) collision with RT and non-RT packets from the other nodes. To resolve this problem, we designed, implemented, and evaluated adaptive traffic smoothing. Specifically, a traffic smoother is installed between the UDP or TCP/IP layer and the Ethernet MAC layer, and works as an interface between them. The traffic smoother first gives RT packets priority over non-RT ones in order to eliminate contention *within* each local node. Second, it smooths a non-RT stream so as to reduce collision with RT packets from the other nodes. This traffic smoothing can dramatically decrease the packet-collision ratio on the network. The traffic smoother, installed at each node, regulates the node's outgoing non-RT stream to maintain a certain traffic-generation rate. In order to provide a reasonable non-RT throughput, the traffic-generation rate is allowed to adapt itself to the underlying network load condition. This traffic smoother requires only a minimal change in the OS kernel *without* any modification to the current standard of Ethernet MAC protocol or the TCP or UDP/IP stack.

We have implemented the traffic smoother on both the Linux and the Windows NT platforms, demonstrating significant reduction of the RT packet deadline-miss ratio when both RT and non-RT packets are transported over the same Ethernet. More precisely, installation of the proposed traffic smoother on every node is shown to reduce the RT message deadline-miss ratio by two orders of magnitude under a heavily-loaded condition, while lowering the non-RT throughput only by half.

Index Terms — 1-persistent CSMA/CD, Ethernet, real-time communication, traffic smoothing.

The work reported in this paper was supported in part by General Motors, Warren, MI, and Lawrence Livermore Laboratories under the TIMES Phase III contract.

1 Introduction

Advances in high-speed network technology have made it possible to transport various application traffic over data communication networks, resulting in an explosive growth of the Internet. The growth of the Internet is creating a huge market for numerous network products related to ATM, FDDI, Ethernet, and so on. Such commercial off-the-shelf (COTS) network products are expanding their application domains. For example, the manufacturing automation industry has been pursuing the use of COTS network products for transporting control messages between PLCs (Programmable Logic Controllers). Traditionally, proprietary networks such as Allen-Bradley's RIO (Remote Input/Output) Network have been used in factory automation to meet the control applications' stringent real-time requirements and deal with harsh working environments. However, the low price and the proven stability of COTS networks have made them attractive for automated manufacturing. Although various high-speed networks like ATM and FDDI are available, Ethernet has been drawing significant interests because of its extremely low price, maturity, and stability proven through its wide deployment and acceptance. Despite its popularity and low-cost, Ethernet has a serious drawback when carrying real-time control messages. In an Ethernet LAN, packets transmitted from different nodes may collide with each other. The medium access control (MAC) protocol of Ethernet, CSMA/CD (Carrier Sense Multiple Access with Collision Detection), allows such collisions. These potential collisions make it impossible to guarantee predictable delays in delivering packets to the local nodes.

In [1], we showed the feasibility of building a real-time control network using Ethernet by installing a traffic smoother at each local node. A traffic smoother regulates the intrinsically bursty packet stream relayed from the UDP or TCP/IP layer, making the packet stream as smooth as possible in order to reduce the chance of packet collisions. By assuming that the smoothed traffic follows a Poisson arrival process, we modeled the CSMA/CD protocol with an Exponential Binary Backoff strategy as a semi-Markov process, and derived the relationship between packet delay distribution and network utilization. Based on the results obtained, we were able to provide a statistical bound on the deadline-miss ratio over Ethernet by keeping the network utilization under a certain limit, called the *network-wide input limit*. To keep network utilization under the network-wide input limit, we assigned a portion of the network-wide input limit to each local node, and made each local node limit its packet generation rate under its assigned portion. We called the node's portion of the network-wide input limit the *station input limit* and installed a traffic smoother at the node to enforce it. Through an experimental study, we demonstrated the effectiveness of the traffic smoothing approach in providing soft real-time

communication over Ethernet. This traffic smoothing approach, however, was inflexible and hence unscalable for the following reason. In that approach the network-wide input limit is fixed once we are given packet deadline and tolerable packet-loss (or deadline-miss) ratio. So, the station input limits must be reduced as the number of local nodes increases within the same LAN. The smaller the station input limit gets, the smaller throughput provided to non-RT traffic. (Note that real-time traffic is not affected, as only non-RT traffic is smoothed [1].) Non-RT packets may experience very large delays when a very small station input limit is assigned to a local node, as discussed in the next section.

In this paper, we propose an *adaptive* traffic smoothing approach to overcome the scalability problem of the approach in [1]. By allowing each local node to vary its maximum traffic-generation rate depending on the current network load, the proposed approach improves its scalability significantly. Apart from this modification, the proposed approach shares the same traffic-smoothing mechanism with the approach in [1]. The traffic smoother is implemented as an interface between the transport layer and the Ethernet MAC layer. This implementation minimizes the modification in the current standard network protocol. We implemented this adaptive traffic smoother on both the Linux and the Windows NT platforms, built testbeds, and conducted extensive experimental studies. Through these experimental studies, we show that the adaptive approach provides much higher throughput for non-RT packets than the non-adaptive scheme in [1], while still providing good delay characteristics for RT packets.

Most earlier work in the area of supporting real-time communication over Ethernet focused on modifying the Ethernet MAC sub-layer so that a bounded channel-access time may be achieved, thus making hard real-time communication possible [2–4]. These approaches are very costly compared to the widely-used current Ethernet standard. Venkatramani and Chiueh [5] proposed implementation of a virtual token ring over Ethernet in order to avoid packet collision. On top of the CSMA/CD protocol, they implemented a token-based medium access control protocol. Thus, their approach does not require modifying any hardware but adds new protocol software. Specifically, it requires significant modification of the OS kernel. Since token management requires a number of functionalities, *e.g.*, restoration of a lost token, it may overload the OS. Our traffic smoothing approach does not require any new MAC protocol but rather installs an interface between the transport layer and the Ethernet MAC layer. The only new function of the interface is to regulate the packet stream, and thus, it is simple to implement.

Another way to bound the channel-access time is to use full duplex Ethernet switches such as IEEE 802.1p or IEEE 802.12, known as Ethernet 100VG-AnyLAN [6], instead of ordinary shared Ethernet

hubs. They both avoid packet collisions by eliminating the CSMA/CD MAC protocol, and thus, can provide bounded packet-delivery delays while retaining compatibility with 10Base-T technology. In particular, 100VG-AnyLAN can provide prioritized service to real-time packets by employing two priority queues. However, both full duplex switched Ethernet and 100VG-AnyLAN are far more expensive than shared Ethernet LANs. At present their prices are an order of magnitude higher than shared Ethernet LANs. In an automated factory, because the traffic-generation rate of each station is, in general, quite low compared to the link capacity, it is not economical to assign a pair of ports of a full-duplex Ethernet switch to each individual control station. In most cases, an Ethernet switch is likely to be used to partition a large-scale LAN into multiple sub-LANs, each of which consists of a shared Ethernet LAN. In this environment, one must still be able to control the traffic arrival behavior of each sub-LAN in order to control end-to-end packet delays through the Ethernet switch. By employing our traffic smoothing mechanism at each sub-LAN, we can control the traffic arrival behavior at each individual sub-LAN, and thus can control the end-to-end delay characteristics in such a switched Ethernet.

The rest of the paper is organized as follows. Section 2 describes our previous traffic-smoothing approach in [1], discusses its scalability problem, and presents the adaptive traffic smoothing approach in procedural forms which we use to implement the traffic smoother on the Linux OS. Sections 3 details our experimental study with the Linux version of the traffic smoother. In Section 4, we describe the implementation of the traffic smoother on the Windows NT, and present the result of its experimental evaluation. The paper concludes with Section 5.

2 Problem Statement and Approach

In an automated manufacturing facility — a prototypical real-time control system — real-time control messages need to be generated and exchanged among the stations in the factory. Most proprietary control networks scan and send I/O (Input/Output) continuously, even though the data changes infrequently or slowly. An alternative approach is to send data only when the data has changed. An event-driven approach for factory automation control messaging is acceptable provided the underlying network can guarantee timely delivery of the updated data. Although the former approach is used in most proprietary control networks, the latter approach is drawing considerable interests since it reduces the rate of generating real-time messages. In this paper, we assume that control stations employ this event-driven approach in generating real-time control messages. In this approach, each control station generates at most one maximum-sized (1500 bytes) IP datagram once every several seconds (or several

hundred milliseconds), and hence, its rate of generating real-time control messages is very low relative to the Ethernet link capacity. Moreover, control messages arrive pseudo-periodically due to the characteristics of the underlying control system. For example, in an automated manufacturing system, control messages notify the end of an operation and the initiation of a new operation to a neighboring station, and these operations are performed periodically.

Concurrently with RT control messages, bursts of non-RT traffic are generated on an irregular basis by controllers and the central server, mainly for the purpose of monitoring production status and downloading programs or new setup parameters. While per-message delay is an important QoS (Quality-of-Service) parameter for real-time applications, average throughput is also important to non-RT traffic. In other words, while a small delivery delay is desirable for non-RT messages, it is *not* a requirement. Because of its burstiness, the arrival rate of non-RT traffic can be quite high during the transmission even if its long-term average traffic arrival rate is low. For example, when only a single station transmits a large burst of non-RT traffic (e.g., a file transfer) over an Ethernet LAN, the traffic arrival rate can reach up to 8–9 Mbps. Such temporarily high network utilization makes it very difficult to provide bounded delivery delays for the other stations' RT messages when both RT and non-RT messages are concurrently transported over the same Ethernet LAN. During the transmission of a large burst of non-RT traffic from another station (node), RT messages may experience a large delay because of collisions and possibly due to the capture effect [7].

Since our previous analysis [1] to resolve this problem was based on the boundedness of network utilization and the assumption that the arrival process is Poisson, it is crucial to employ a traffic smoother at every local node. By making each local node keep its traffic-generation rate under its station input limit, one can keep network utilization under the network-wide input limit. We installed the traffic smoother between the TCP/IP layer and the Ethernet MAC (Medium Access Control) layer, as shown in Figure 1, in order to minimize the changes in the current standard protocol stack while achieving the good smoothing effect. Although installing the traffic smoother on top of the TCP/IP layer would be a simpler approach, the resulting smoothed packet stream would be distorted (become burstier) due to the un-smoothed TCP/IP protocol messages in the packet stream that do not convey application data.

When a burst of non-RT messages arrive from the TCP/IP layer, the traffic smoother spreads them out by enforcing a minimum packet inter-arrival time at the Ethernet MAC layer to meet the station input limit. More specifically, the traffic smoother regulates the packet stream using a credit bucket, which is the same as the well-known leaky-bucket regulator [8]. The credit buffer has two parameters:

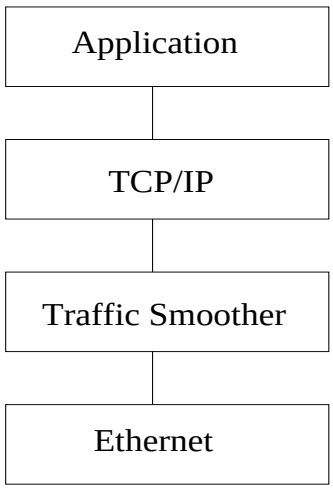


Figure 1: The software architecture

credit bucket depth (CBD) and refresh period (RP). CBD limits the maximum number of credits that can be stored in the credit bucket. Up to CBD credits are added to the bucket every RP seconds. If the number of credits exceeds CBD, overflow credits are discarded. When a packet (IP datagram) arrives from the IP layer, if there is at least one credit in the bucket, the traffic smoother forwards it to the Ethernet NIC (Network Interface Card) and removes as many credits as the size of the packet (in bytes). When the number of available credits is smaller than the packet size, credits are allowed to be "borrowed." So, the balance of credits can be negative. If there are no credits in the credit bucket, the packet is held in the buffer until one or more credits become available. By changing RP and CBD, one can control the burstiness of a packet stream while keeping the same average throughput guarantee. For example, if we set $\frac{CBD}{RP}$ to 312500, the average throughput guaranteed for a station is 312.5 Kbytes/sec or 2.5 Mbps. Two possible pairs of (CBD, RP) satisfying the ratio are (1500, 0.0048) and (150000, 0.48). When (CBD, RP) = (1500, 0.0048), the maximum amount of traffic that can be transmitted consecutively is limited to 2999 bytes (1499 bytes plus 1500 bytes). In this case, traffic is smoothed with a very fine time granularity and the worst-case traffic arrival rate in a short period is the same as the average traffic arrival rate. In the other case, up to 151499 bytes can be transmitted consecutively. Although the average traffic arrival rate is the same, this case generates a much burstier output and the worst-case traffic arrival rate in a short period is much higher than the average traffic arrival rate. Our experimental study has shown that better real-time performance can be achieved with finer-granularity smoothing.

In this approach, within a local node, RT packets are given priority over non-RT packets, and only

non-RT packets are delayed to keep the station traffic-arrival rate (which includes both RT and non-RT traffic) under the station input limit. That is, transmission of extra RT packets causes non-RT packets to experience additional delays. RT traffic is assumed to arrive pseudo-periodically and thus, is already smooth as discussed earlier.

The network-wide input limit can be either equally distributed among local nodes or disproportionately distributed depending on each local node's needs. Once the station input limit is assigned to a local node, the maximum traffic transmission rate of the node is fixed at its station input limit by the traffic smoother. We call this type of traffic smoothing *fixed-rate traffic smoothing* and the traffic smoother is called a *fixed-rate traffic smoother*. Since, however, the station input limit is calculated based on the worst-case traffic arrival scenario in which all the local nodes are generating traffic at their maximum allowable rates, it depends on the number of nodes, more precisely, the maximum number of nodes that may generate non-RT traffic. This raises a scalability issue. Especially, in a real-time control network in which all the nodes do not always generate non-RT traffic concurrently, non-RT IP datagrams may experience excessively large delays even when overall network utilization is low. That is, when only a few of the nodes are generating non-RT traffic during a certain time period, the bandwidth assigned to the rest of the nodes is wasted and non-RT IP datagrams experience unnecessarily large delays.

We propose a new traffic smoothing approach to resolve the poor scalability of fixed-rate traffic smoothing while still providing low delays for RT messages. In the next two subsections, we describe the new traffic smoother. In particular, in Section 2.2, we present the traffic smoother in procedural forms which is used to implement the traffic smoother in the Linux OS. The Windows NT version of the smoother will be presented in Section 4.2.

2.1 Adaptive-Rate Traffic Smoothing

In order to meet the delay requirement of RT packets, we still need to regulate non-RT traffic as smoothly as possible and keep network utilization under a certain limit. Unlike a fixed-rate traffic smoother, however, our new traffic smoother, called an *adaptive-rate traffic smoother*, changes the station input limit at each local node depending on the current network traffic arrival rate. That is, if network utilization by non-RT traffic is low, those nodes generating non-RT traffic are allowed to increase their station input limits subject to the condition that the overall network utilization does not cause RT packets to experience delays larger than those in the fixed-rate traffic smoothing approach.

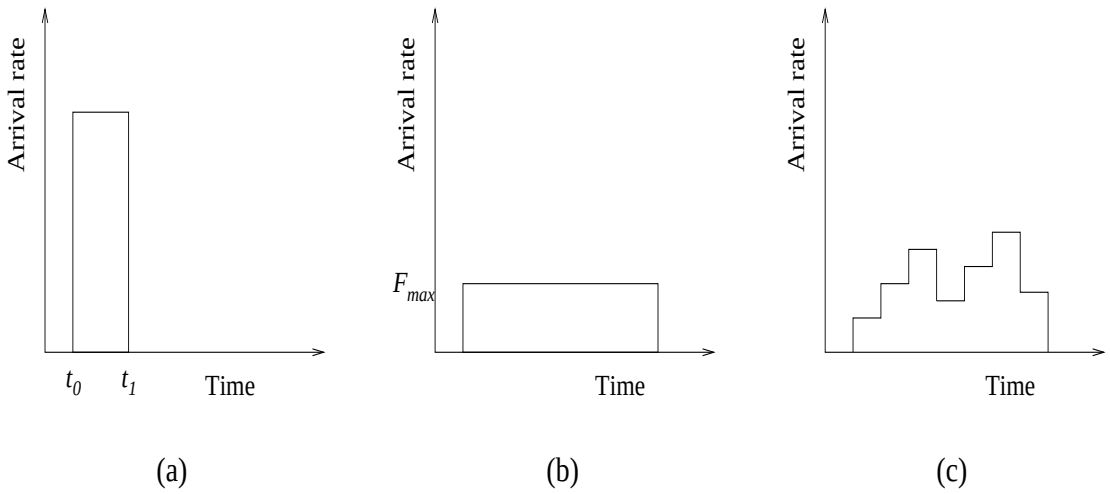


Figure 2: Traffic smoothing

Likewise, as network utilization by non-RT traffic gets higher, those nodes generating non-RT traffic lower their station input limits. Figure 2 compares the arrival rate of traffic smoothed by a fixed-rate traffic smoother and that by an adaptive-rate traffic smoother. Let's assume that a large burst of packets generated by an application through the TCP/IP layer during $[t_0, t_1]$. If the node had a large TCP window when the burst has arrived from the application, the traffic arrival duration may be very short and the arrival rate may be quite high as shown in Figure 2(a). The arrival rate of the traffic smoothed by a fixed-rate traffic smoother is shown in Figure 2(b). Here, the station input limit is set to F_{max}, and thus, the arrival rate is always kept under F_{max}. On the other hand, the arrival rate of traffic smoothed by an adaptive-rate traffic smoother is a piece-wise constant function of time as shown in Figure 2(c), and it depends on the traffic-generation statistics of the other nodes. That is, the adaptive-rate traffic smoother regulates the packet stream using the station input limit in order to keep the stream as smooth as the fixed-rate one does, but the station input limit changes with time.

In order to implement an adaptive-rate traffic smoother which meets the delay requirement of RT packets while providing improved average throughput for non-RT packets, we must resolve the following two problems: (1) how to detect a change in network utilization and (2) how to adapt to the detected change. An efficient detection mechanism is essential for the adaptation to be fast enough to meet the delay requirement of RT packets. However, since unlike ATM or FDDI, the CSMA/CD protocol is not a reservation-based medium access control scheme, direct information on the current network utilization is unavailable to local nodes. Therefore, each local node must depend on an indirect method of determining network utilization such as detecting packet collisions at its NIC or measuring

the buffer-clearing rate at its Ethernet device driver. Or, each local node may use the promiscuous mode to measure the network utilization for a recent period of time. We have chosen the first option for its good responsiveness. In particular, if the traffic smoother is set to vacate the credit buffer immediately upon detection of a collision, transmission of non-RT packets is suspended, except for those packets already in NICs. This increases the chance to deliver the RT packets generated from other nodes sooner, as they do not suffer the "packet starvation" [9] caused by the burst of non-RT packets generated from this node. For this reason, we use packet collision as a trigger to decrease throughout as well as to deplete the current credits.

2.2 Harmonic-Increase and Multiplicative-Decrease Adaptation

Next, let's consider the adaptation mechanism. Two parameters, CBD and RP, can be used to change the station input limit which is given as $\frac{CBD}{RP}$. By changing CBD while keeping RP constant, we can change the station input limit, but this approach causes the size of a burst to fluctuate. Since we want to keep the packet stream as smooth as possible, we instead vary RP while keeping CBD constant. Especially, by setting CBD to the Ethernet MTU (Maximum Transfer Unit) (i.e., 1500 bytes), one can set the maximum amount of traffic that can be transmitted up to 2999 bytes. In this approach, one can increase the station input limit by decreasing RP, and vice versa.

There are many ways to change the station input limit. For example, one may employ the "slow-start increase and multiplicative decrease" that is being used in the TCP/IP congestion avoidance mechanism [10]. In this paper, we use a very simple adaptation mechanism called *Harmonic-Increase and Multiplicative-Decrease Adaptation* (HIMD). HIMD is similar to the slow-start increase and multiplicative-decrease algorithm in decreasing the throughput but *differs* in increasing the throughput. HIMD works as follows. First, HIMD periodically increases the station input limit by decreasing RP periodically in the absence of packet collisions. The size of each decrement is fixed at a constant, and thus, the station input limit is harmonically incremented. This harmonic increment is conservative but easy to implement. When a packet collision is detected, the traffic smoother immediately depletes the current credits, delays the transfer of the non-RT packet, and doubles RP. By choosing an appropriate size of decrement for RP, one can adapt the station input limit very fast.

Figures 3 and 4 describe the traffic smoother in a procedural form. First, the procedure *smoothing* in Figure 3 smoothes the packet stream. To provide low delivery delays to RT packets, the traffic smoother maintains a priority queue with two priority levels. The high-priority queue is used for

storing RT packets and the low-priority queue is used for storing non-RT packets. When packets arrive from the upper layer, they are inserted into the corresponding priority queue in the order of their arrivals. Whether a packet is RT or non-RT is determined using the Type-of-Service (ToS) field of an IP datagram. When the procedure *smoothing* is called by the kernel scheduler, it first checks whether there is a packet waiting to be transferred to NIC in the queue, starting from the high-priority queue. If there is a packet belonging to a real-time stream, it is immediately transferred to NIC by a function call, *send_to_NIC*, and as many credits, denoted by *CurrentNetworkShare*, as the size of the packet, denoted by *Packet.FrameSize*, are removed from the credit bucket. If the packet belongs to a non-RT session, the last collision time, denoted by *LastCollisionTime* (which will be described shortly), is checked. If the difference between the current time and the last collision time falls within a certain bound α, the traffic smoother assumes that another station is trying to send a real-time or non-real-time packet. Therefore, it returns the packet to the low-priority queue by making a function call, *send_back_to_queue*. In addition, it vacates the credit bucket by setting *CurrentNetworkShare* to zero, and doubles RP. RP is capped by RP_{max}. If the packet belongs to a non-RT session and *CurrentNetworkShare* is positive while there has been no recent collision, the packet is transferred to NIC by a function call, *send_to_NIC*, and *CurrentNetworkShare* is decremented by the size of the packet. If there is no credit in the credit bucket, i.e., *CurrentNetworkShare* ≤ 0, the packet is returned to its original location in the low-priority queue. The packets sent back to the low-priority queue are served the next time when this procedure is called by the kernel scheduler.

Procedure *refresh* in Figure 4 is called once every τ where τ is a user-defined parameter, and Procedure *refresh* decrements RP by Δ (harmonic decrease). The minimum value of RP is set to RP_{min}. In addition, when the current time reaches $NextRefreshTime$, it increments the number of credits by CBD, and sets the next credit bucket refreshing time to $CurrentTime + RP$. If the total number of credits exceeds CBD, the number of credits is set to CBD.

In addition to the above two procedures, we need to modify the Ethernet device driver to record the time when a packet in NIC encounters a collision so that the procedure *smoothing* may use it. Many available device drivers request their Ethernet NICs to notify the number of collisions that the recently-transmitted packet has experienced. If this function is not provided by default, one should modify the device driver to invoke it. When the device driver receives this collision information, it records the current time as $LastCollisionTime$ if the recently-transmitted packet has experienced a collision.

In our traffic smoother, α, Δ, τ, RP_{max}, and RP_{min} are user-controllable parameters. By using

```
Procedure smoothing
If (Packet.TypeOfService = RealTime) then {
        send_to_NIC;
        CurrentNetworkShare := CurrentNetworkShare − Packet.FrameSize;
      }
else if (LastCollisionTime ≥ CurrentTime − α) then {
        send_back_to_queue;
        CurrentNetworkShare := 0;
        RP = min(RP_max, 2 × RP);
      }
else if (CurrentNetworkShare > 0) then {
        send_to_NIC;
        CurrentNetworkShare := CurrentNetworkShare − Packet.FrameSize;
      }
else send_back_to_queue;
```

Figure 3: Procedure of traffic smoothing

different values, one can obtain different delay and throughput characteristics.

The idea of adapting the traffic-generation rate has already been implemented in other protocols in order to avoid network congestion and improve throughput. For example, the TCP congestion avoidance algorithm and the Ethernet collision resolution protocol (Exponential Binary Backoff) have already been in use. Our scheme lies between them in time scale, but shares the same basic idea and goal — avoid network congestion — with them. One significant difference is that our scheme works only on non-RT traffic to provide better delay characteristics to RT traffic.

3 Experimental Evaluation on Linux

In this section, we present the experimental evaluation results on a testbed of Linux workstations. We installed the adaptive-rate traffic smoother at all the local nodes, and measured the delay characteristics of RT messages while measuring the throughput of non-RT messages. In addition, we conducted similar experiments with the fixed-rate traffic smoother and without employing any traffic smoothing mechanism for the purpose of comparison.

Procedure $refresh$

$RP := \max(RP_{min}, RP - \Delta)$;

if $(CurrentTime = NextRefreshTime)$ **then** {

$CurrentNetworkShare := \min(CurrentNetworkShare + CBD, CBD)$;

$NextRefreshTime := CurrentTime + RP$;

}

Figure 4: Procedure of refreshing parameters

3.1 The Environment

The Linux testbed consists of two 300 MHz Intel Pentium II PCs, five 75 MHz Pentium laptop computers, and four 486 DX/4 laptop computers, and they are connected through a 10BASE-T Ethernet LAN. The collision domain diameter is 10 m. We configure the local nodes as PC-1 — PC-10 and a monitoring station. Figure 5 shows the topology of our testbed. One 300 MHz Intel Pentium II PC works as the monitoring station, and since our target application is automated factory networking, the rest of the PCs simulate PLCs. We use TCP sockets for transmitting RT control messages as well as non-RT messages[1]. The PCs exchange real-time control information with RT messages. More specifically, PC-1 sends a 100 byte long RT control information which is contained in a high-priority IP datagram to PC-2. Then, PC-2 echos back to PC-1 with a high-priority IP datagram of the same size. Likewise, PC-n and PC-$(n+1)$ exchange RT control information of the same size where $n = 1, \ldots, 9$. PC-10 sends a RT control message to PC-1, and PC-1 echos back to PC-10. We made the inter-arrival time of real-time control messages at each simulated PLC follow an exponential distribution, and set the average message inter-arrival time to 0.3 sec. Since we must count both RT control and echo messages, the network load due to RT messages is $(2 \cdot 100 \cdot 8 \cdot 10/0.3)$ bps, i.e., 53.3 kbps[2]. The traffic-generation rate was chosen to reflect the low traffic condition observed in most automated manufacturing facilities.

In addition to RT messages, PCs generate non-RT messages when the monitoring station requests

[1] We recognize that UDP sockets are generally preferred for transporting RT control messages in order to avoid possible delays due to the TCP data-loss recovery mechanisms. Although we employed TCP sockets for transporting RT control messages, we did not observe any data-loss recovery delays in our experiments.

[2] This is the network load seen by the application layer. In the Ethernet physical layer, the load is slightly higher than this value since the TCP/IP header, the Ethernet MAC header and the framing field must be counted towards the total data size. For non-RT traffic, we also measure the network load from the standpoint of the application layer.

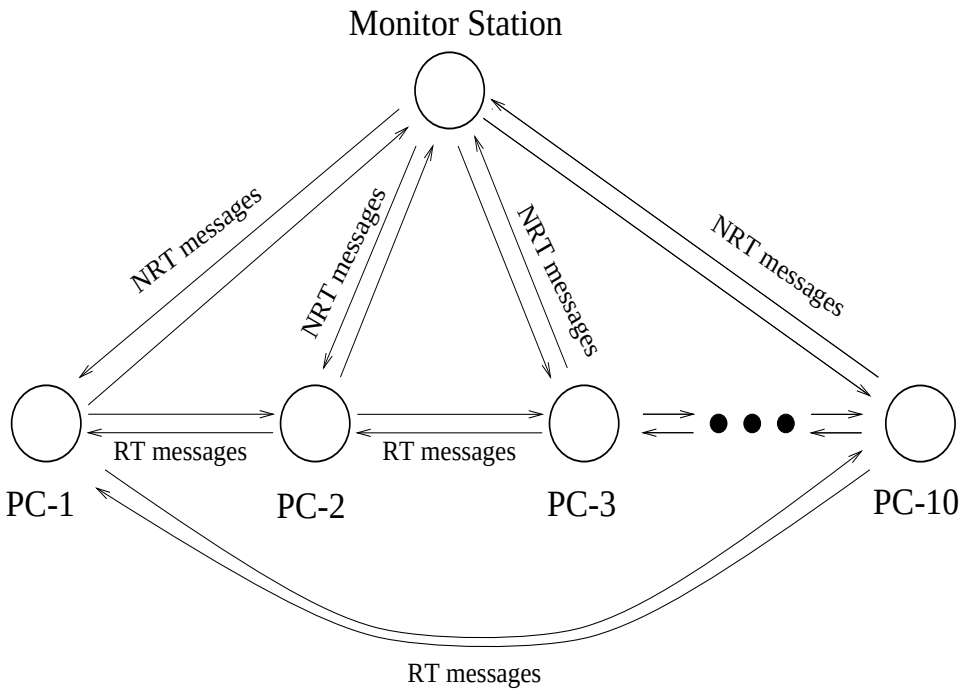


Figure 5: Experiment environment

them to send their status information. The size of non-RT traffic generated by an application is 1 Mbytes, and it is transmitted as a sequence of low-priority IP datagrams. This results in a high instantaneous traffic-generation rate (i.e., a burst of non-RT messages) especially at the TCP/IP layer.

To investigate the effectiveness of the adaptive-rate traffic smoother, we measured the roundtrip delay of every RT control message and the time to transmit each non-RT burst while transporting both types of traffic over the Ethernet and varying the non-RT traffic-generation rate. From these measurements, we calculated the deadline-miss ratio of RT messages and the average time to transmit a 1 Mbyte-long non-RT burst. We set the roundtrip deadline of RT messages to 129.6 msec[3]. Since a real-time message is considered lost if its deadline is missed, we treated the deadline-miss ratio as the message-loss ratio.

We conducted two sets of experiments with different non-RT traffic-generation scenarios. In the first set which we call *non-greedy mode*, the non-RT burst inter-arrival time of an activated[4] PC was exponentially-distributed, and the average burst inter-arrival time was set to 25 sec. Then, the

[3] This value was selected through an analysis shown in [1]. According to [1], when 10 transmission trials are allowed before a messages is declared lost, the worst-case delay is 64.8 msec. We simply doubled that value to select a worst-case roundtrip delay of 129.6 msec as the roundtrip deadline.

[4] A station/PC is said to be *activated* if the monitoring station requested its status information.

average non-RT traffic-generation rate of an activated PC is $10^6 \times 8/25 = 320$ kbps. By changing the number of activated PCs, one can control the non-RT traffic load. When 10 PCs are activated, the network load of non-RT traffic is 3.2 Mbps, i.e., approximately[5] 32% of the Ethernet capacity. In the second set which we call *greedy mode*, an activated PC was set to generate non-RT bursts in succession. That is, once it had finished the transmission of a non-RT burst, an activated PC starts transmission of the next burst immediately. In this scenario, the network can be overloaded even with a single activated PC. In reality, however, the maximum achievable network utilization is about 0.75 because of the congestion-avoidance mechanism of the TCP flow control and the Ethernet collision-resolution mechanism.

3.2 Results

In the non-greedy mode, we experimented with the adaptive-rate traffic smoothing, the fixed-rate traffic smoothing, and without any traffic smoothing at all. In each case, we varied the number of activated PCs from 2 to 10, i.e., we changed the non-RT traffic load from 0.064 to 0.32, and measured the loss ratio of RT messages and the average time to transmit a 1 Mbyte-long non-RT burst. The total number of RT messages generated in each case was 500,000, which dictates the confidence interval of the deadline-miss ratio of RT messages. The results are plotted in Figures 6 and 7. Figure 6 shows the RT message-loss ratio, i.e., the deadline-miss ratio, and Figure 7 shows the average time to transmit a 1 Mbyte-long non-RT burst. In the absence of traffic smoothing, like in a conventional Ethernet LAN, the measured deadline-miss ratio of RT messages ranged from 2.356×10^{-3} to 1.682×10^{-2}, and the maximum length of 99 % confidence intervals was 1.33×10^{-6}. Compared to the traffic-smoothing schemes, the case of no traffic smoothing resulted in high deadline-miss ratios. On the other hand, it showed the smallest average transmission times of non-RT bursts among the three schemes as shown in Figure 7. They ranged from 1.073 to 1.819 sec, meaning that the average throughput provided to each activated PC for transmitting non-RT bursts ranged from 4.398 to 7.455 Mbps.

To evaluate the fixed-rate traffic smoothing, we installed a fixed-rate traffic smoother at every PC. CBD was set to 1500 and RP was set to 3.6 msec. Thus, the minimum and maximum throughputs provided to each PC for transmitting non-RT bursts were 0.33 Mbps. As a result, it took about 25 sec for each activated PC to transmit a non-RT burst regardless of the number of activated PCs as shown in Figure 7. The delay characteristics of RT messages were significantly improved. As shown

[5] since we should consider the headers and framing fields for the physical layer's traffic-generation rate

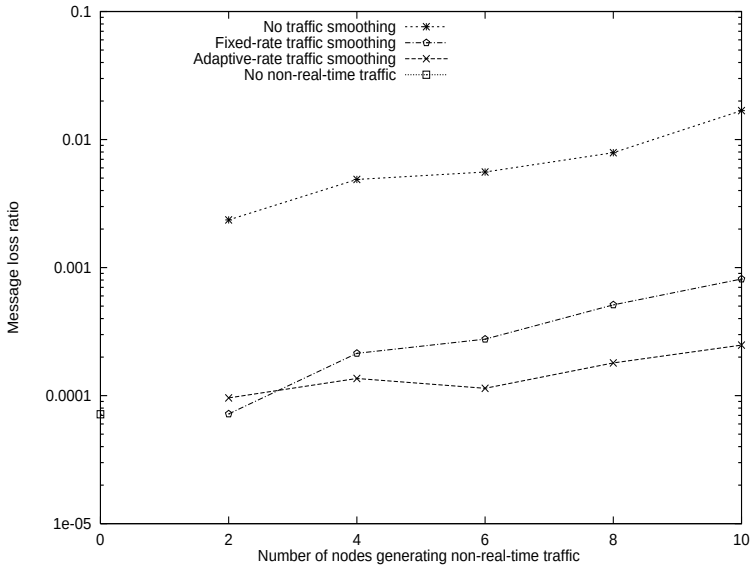


Figure 6: RT message loss ratio in the non-greedy mode

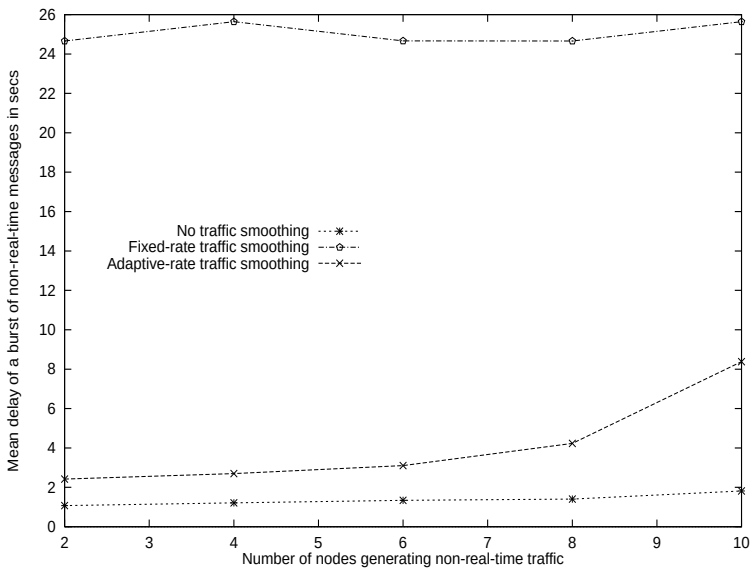


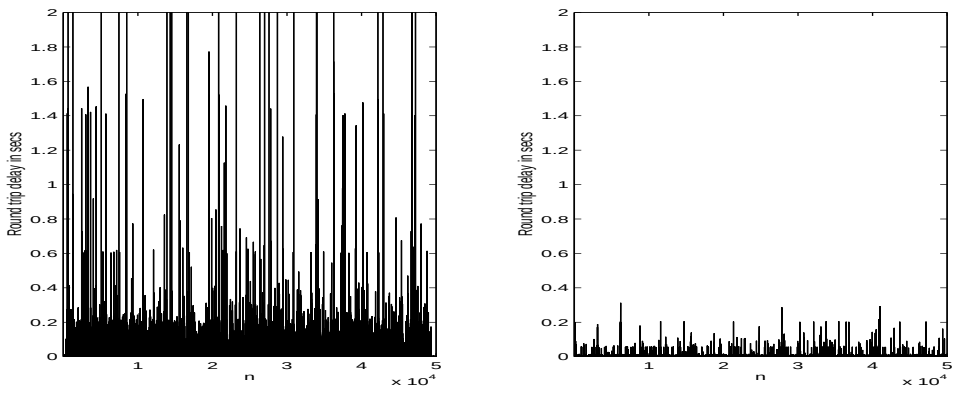
Figure 7: Average transmission time of a 1 Mbyte non-RT burst in the non-greedy mode

in Figure 6, the deadline-miss ratio of RT messages[6] was kept under 10^{-3}, although it increased with the number of activated PCs.

To evaluate the effectiveness of adaptive-rate traffic smoothing, we installed an adaptive-rate traffic smoother at every PC. The parameters were chosen as: 10 msec for α, 100 μsec for Δ, 100 msec for RP_{max}, and 3 msec for RP_{min}; and τ was set to 1 msec. Thus, the decrement rate of RP was 0.1, and the maximum throughput that can be provided to each PC in a short-term was 4 Mbps. These values were selected empirically to achieve a low deadline-miss ratio of RT messages without sacrificing the throughput provided to non-RT bursts. Compared to the cases of no traffic smoothing and fixed-rate traffic smoothing, adaptive-rate traffic smoothing showed the smallest deadline-miss ratio, except when only two PCs were activated. For the case of two activated PCs, adaptive-rate traffic smoothing showed a slightly larger value than fixed-rate traffic smoothing as shown in Figure 6. The maximum deadline-miss ratio of adaptive-rate traffic smoothing was 2.48×10^{-4}, which is smaller than that (1/3) of fixed-rate traffic smoothing and is attained when all the PCs were activated. The small deadline-miss ratio of adaptive-rate traffic smoothing is pronounced particularly when it is compared against the other schemes in terms of the throughput provided to non-RT traffic which is shown in Figure 7 as a form of delay. The average transmission time of a 1 Mbyte non-RT burst ranged from 2.42 to 8.38 sec, and thus, the average throughput provided to an activated PC for non-RT traffic ranged between 0.955 and 3.36 Mbps which is much larger than 0.33 Mbps, the throughput provided to non-RT traffic in the case of fixed-rate traffic smoothing. In addition, the throughput provided to non-RT traffic in the case of adaptive-rate traffic smoothing increased as the number of activated PCs decreased, unlike the result obtained in the fixed-rate traffic smoothing case. This indicates that the adaptive-rate traffic smoothing does not waste the bandwidth when a small number of PCs are activated, thus overcoming the scalability problem, as we argued in the previous section.

In addition to the three traffic smoothing schemes, we conducted an experiment without activating any PC. In this case, it does not matter what traffic smoothing scheme is enforced since there is no non-RT traffic to be smoothed. The total traffic arrival rate which was due only to real-time traffic sources, was 53.3 kbps as mentioned earlier. The deadline-miss ratio thus obtained was 7.15×10^{-5}. One can see that the result of adaptive-rate traffic smoothing is very close to this value. This indicates that the delay characteristics of real-time messages is almost unaffected by the presence of non-RT messages in the adaptive-rate traffic smoothing as compared to the other two schemes. This is due to

[6]The lengths of the confidence intervals were smaller than 10^{-6}. The lengths of the confidence intervals in all the cases hereafter were kept smaller than 10^{-5}.



(a) (b)

Figure 8: Roundtrip delay sequences of real-time messages: (a) no traffic smoothing (b) adaptive-rate traffic smoothing

the fact that the smoother stops transmitting non-RT traffic and doubles its RP as soon as it finds the on-going transmission experiencing any collision with a real-time message or a non-RT message transmitted by another PC.

Figure 8 illustrates the effectiveness of adaptive-rate traffic smoothing in achieving soft RT guarantees over Ethernet. Figures 8(a) and 8(b) show, respectively, the roundtrip delay sequences of 50,000 RT messages when no traffic smoothing was enforced and when the adaptive-rate traffic smoothing was enforced. The number of activated nodes were 10, and thus, the non-RT traffic load was 0.32. As shown in Figure 8(a), when no traffic smoothing was applied, a fair number of real-time messages experienced roundtrip delays larger than 1 sec while the roundtrip delays were kept below 300 msec when the adaptive-rate traffic smoothing was enforced. When the fixed-rate traffic smoothing was enforced, the measured roundtrip delay sequence of real-time messages was similar to that shown in Figure 8(b).

The greedy-mode experiments re-confirmed the effectiveness of adaptive-rate traffic smoothing. In this case, an activated PC was allowed to transmit non-RT traffic at its maximum capacity without being restricted by an application-level flow control. Since the worst-case network-wide non-RT traffic arrival rate in this environment is deterministic when the fixed-rate traffic smoothing is enforced, we only compared the case without traffic smoothing with that with adaptive-rate traffic smoothing.

When no traffic smoothing is enforced, the network can be fully-loaded by only one activated PC, causing extremely large delays to RT messages. Even though real-time IP datagrams are given priority

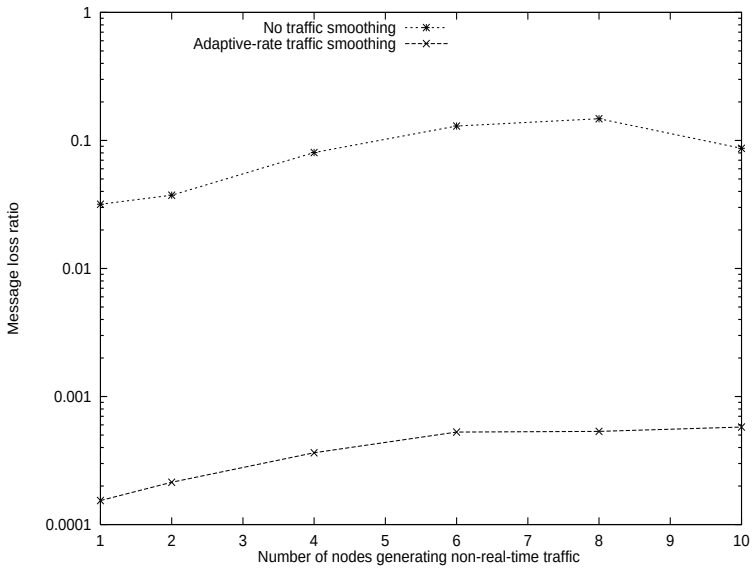


Figure 9: RT message loss ratio in the greedy mode

over non-RT ones within the same PC, they may collide with non-RT datagrams transmitted from the other PCs and experience large delays. Under such an extreme traffic-arrival condition, adaptive-rate traffic smoothing proved to work remarkably well. Figures 9 and 10 show the experimental results. Figure 9 shows the deadline-miss ratios of RT messages for different numbers of activated PCs when no traffic smoothing was enforced and when adaptive-rate traffic smoothing was enforced. Figure 10 shows the throughput provided to all the activated PCs for transmitting non-RT bursts. This was derived from the number of activated PCs and the average transmission time for transmitting a single burst, considering that the throughput provided to an activated PC is given by the burst size divided by the transmission time.

When no traffic smoothing was enforced, the deadline-miss ratios of RT messages were extremely high, i.e., in the range of 10^{-1} as shown in Figure 9, although the throughput provided for non-RT traffic reached up to 0.74 as shown in Figure 10. Thanks to the flow controls mention above, we could not overload the network.

On the other hand, when the adaptive-rate traffic smoothing was enforced on every activated PC, the throughput for transmitting non-RT bursts was reduced, approximately by half, but the RT message deadline-miss ratios dropped dramatically. They ranged from 1.54×10^{-4} to 5.78×10^{-4}, and were much smaller than those achieved in the case of no traffic smoothing. In this environment, the transmission capability of a TCP socket of an activated PC was restricted not only by the TCP

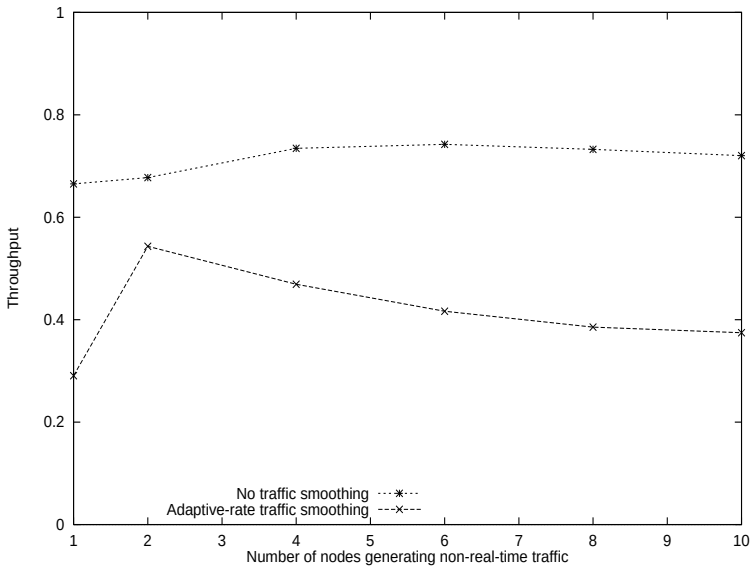


Figure 10: Throughput of non-RT traffic in the greedy mode

flow control and Ethernet collision-resolution mechanism, but also by the adaptation mechanism of the traffic smoother. This explains the lower throughput achieved by the adaptive-rate traffic smoothing. These experimental results indicate that we can build *two virtual networks* — a soft real-time control network and an Ethernet LAN with 5 Mbps transmission capability — using a single Ethernet LAN if the adaptive-rate traffic smoothing is employed.

In addition to HIMD, we experimented with various adaptation mechanism (e.g., Linear Increment and Multiplicative Decrement), but HIMD yielded the best adaptation performance, thus omitting their discussion.

4 Implementation and Experimental Evaluation on Windows NT

We have also implemented the adaptive traffic smoother on the Windows NT platforms, and conducted an indepth experimental evaluation. This implementation was motivated by the fact that the Windows NT is widely deployed and used but gives less leverages to software developers than the Linux OS. Although we will compare the performance of both the NT and the Linux implementations, our focus will be placed on their qualitative aspects since their real-time performance depends not only on the communication protocol stacks, but also on the operating systems themselves. For this reason, we built an NT testbed which is totally independent of the Linux testbed.

4.1 The Traffic Smoother on the Windows NT

Unlike the Linux OS, we are not allowed to, nor do we want to, modify the NT kernel. Moreover, the source codes of Ethernet device drivers are not usually available to end users. Fortunately, however, the Windows NT allows end users to insert an intermediate driver — called the *NDIS* (Network Driver Interface Specification) intermediate driver [11] — between the extant transport protocol layers and network device drivers. Since it is a standard interface between protocol layers, we implemented the traffic smoother as an NDIS intermediate driver without modifying the NT kernel.

The NT-version traffic smoother consists of two procedures: *packet_classifer* and *process_NRTQueue*. Procedure *packet_classifer* simply checks the class[7] of a packet arriving from the IP layer, and inserts it into one of two queues according to its type, real-time or non-real-time queue. Packets inserted into the RT queue are immediately transferred to the Ethernet NIC.

Packets inserted into the non-RT queue are serviced by a timer service routine, *process_NRTQueue*, shown in Figure 11. When this routine is called, it checks whether the non-RT queue is empty or not. If it is not empty, the function $Serve(CBD)$ is called, which transfers at most CBD bytes to the Ethernet NIC.

Upon completion of $Serve(CBD)$, the function $CheckCollisions()$ is called, checking if the most recently-transferred packet has successfully been transmitted over the network without having experienced any collision. Most Ethernet cards and drivers are designed to determine how many packets have experienced collisions at least once before their successful transmission, and this information is made available to upper layers *upon request* in case of the Windows NT. Using this information, $CheckCollisions()$ collects the most recently-transferred packet's collision statistics and indicates the current network congestion condition. However, frequent collection of collision statistics by the traffic smoother will introduce a high overhead on CPU, and can even freeze the system in an extreme case. Therefore, $CheckCollisions()$ should not be called too often. As a result, the NT-version traffic smoother is less responsive than the Linux version which uses the most recent collision-time information as described in Section 2.2. Upon detection of a collision, the traffic smoother decreases its sending rate by invoking the function $Decrease(CBD, RP)$. In contrast with the Linux version, the NT version allows both CBD and RP to be changed, because the timing granularity that can be used in the Windows NT is fixed at 10 msec. That is, Procedure $CheckCollisions()$ can be called at most once

[7] In the Windows NT, the ToS field of an IP datagram cannot be changed using the setsockopt() function like in the Linux. For this reason, we chose the protocol name as a way of differentiating real-time from non-real-time packets. We used UDP sockets for real-time connections and TCP sockets for non-real-time connections.

every 10 msec. If we set the maximum CBD to 1500 as we did in the Linux version, the maximum throughput provided to a single node is 1.2 Mbps, which leads to low network utilization. Therefore, we must allow CBD to be larger than 1500. However, in order to avoid the poor responsiveness of the traffic smoother due to large bursts, we must set a reasonable maximum value on CBD. Specifically, we set the maximum CBD to 4500. Now, when $Decrease(CBD, RP)$ is called, the traffic smoother decrements CBD by 1500 if CBD is greater than 1500. Otherwise, it doubles RP as in the Linux version.

If $CheckCollisions()$ indicates that there was no collision, function $Increase(CBD, RP)$ is called. If RP is larger than 10 msec, RP is decremented by Δ as in the Linux version. The minimum value of RP is set to 10 msec. If RP = 10 msec, CBD is incremented by 1500. RP is capped by RP_{max} which was set to 1000 msec. Δ was set to 10 msec in our implementation.

When the non-RT queue is empty upon invocation of $process_NRTQueue$, function $Increase(CBD, RP)$ is called.

After finishing all the routines, procedure $process_NRTQueue$ sets its next invocation time which is given as the current time plus RP.

```
Procedure process_NRTQueue
If (NonRealTimeQueue != Empty) then {
        Serve(CBD);
        if (CheckCollisions() == True) then {
                Decrease(CBD, RP);
            }
        else {
                Increase(CBD, RP);
            }
    }
else {
        Increase(CBD, RP);
    }
SetNextServiceTime(RP);
```

Figure 11: Procedure of handling non-RT queue

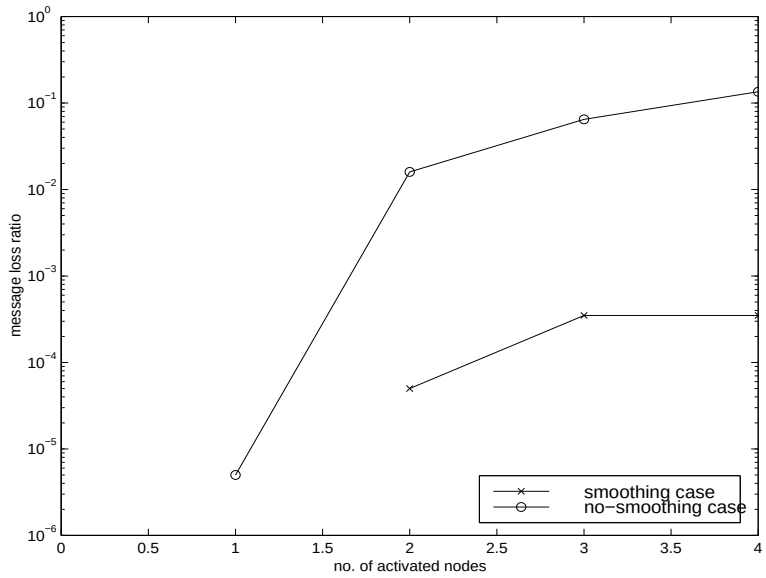


Figure 12: RT message loss ratio in the greedy mode observed in the NT testbed

4.2 Experimental Evaluation

We built a testbed with 5 NT workstations, and conducted the same experiment as we did on the Linux testbed. We now present the results of the greedy-mode experiments.

The NT testbed consists of one 400 MHz Pentium II PC, and four 133 MHz Pentium laptop computers, and they are connected through a 10 BASE-T Ethernet LAN. The collision domain diameter is 10 m. We configured the NT testbed similarly to the Linux testbed as shown Figure 5, except that there are one monitor station (400 MHz Pentium II PC) and four local stations denoted by PC-1, PC-2, PC-3, and PC-4. RT control and echo messages were generated from the local stations in exactly the same way as in the Linux testbed. Therefore, the network load due to RT messages is $2 \times 100 \times 8 \times 4/0.3 = 21.3$ kbps. We set the roundtrip deadline of RT messages to 129.6 msec as in the Linux testbed. The volume of non-RT traffic generated by an application was set to 1 Mbytes, and activated stations generate non-RT traffic in the greedy mode.

Figure 12 shows the RT message loss ratio in the greedy mode. The NT testbed generated experimental results similar to the ones we obtained from the Linux testbed. When the adaptive-rate traffic smoothing was enforced (labeled by "smoothing case" in Figure 12), the RT message loss ratio ranged from 0 to 3.5×10^{-4} when the number of activated nodes changes from 1 to 4. Especially, when a single node was activated, no message was observed to miss its deadline among 400,000 messages. When no

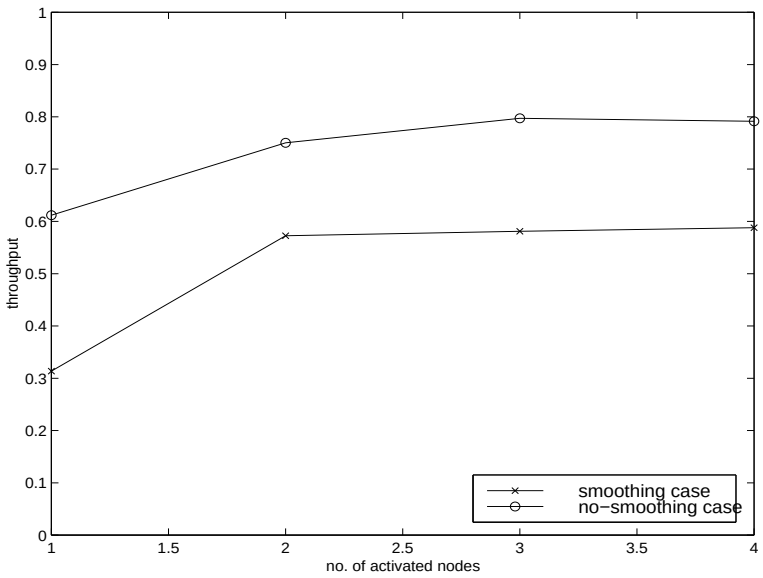


Figure 13: Throughput of non-RT traffic in the greedy mode observed in the NT testbed

traffic smoothing was enforced, RT message loss ratios are two orders of magnitude larger than those observed when the adaptive-rate traffic smoothing was enforced. As the Linux-version (see Figure 9), the NT-version adaptive-rate traffic smoother dramatically drops the RT messages loss ratio.

Figure 13 shows the throughput provided to the non-RT traffic in the greedy mode. Compared to the Linux-version, the NT-version traffic smoother has better non-RT throughput characteristics. That is, when the traffic smoother was enforced, the throughput provided to non-RT traffic drops only by 25 %, approximately. This is possibly due to the smaller number of nodes in the NT testbed, but we did not experiment with a larger testbed yet to confirm this.

Overall, the NT-version adaptive-rate traffic smoother can provide reasonable RT message deadline-miss ratios while providing an acceptable throughput for non-RT traffic. In addition, NT implementation of the adaptive-rate traffic smoother is relatively simple thanks to the NDIS intermediate driver.

5 Conclusion

In this paper, we developed a methodology for providing soft real-time communication services over an Ethernet LAN which transports both real-time and non-real-time packets. To provide faster and predictive delivery service for real-time packets, in each local node real-time packets are given priority

over non-real-time ones. In order to reduce the collision with real-time packets transmitted from other nodes, each local node is required to smooth its non-real-time packet stream. Specifically, we have installed a traffic smoother at each local node between the transport layer and the Ethernet MAC layer. The packet stream arriving from the TCP/IP layer is bursty by nature, and hence, the traffic smoother regulates the stream to be as smooth as possible, then relays the packets to the Ethernet MAC layer. Using traffic smoothing, one can dramatically reduce the packet-collision ratio on the network. Each traffic smoother regulates its packet stream using a certain traffic-generation rate. By allowing the traffic-generation rate to adapt to the current network load condition, we were able to provide reasonably good throughput to non-real-time traffic while meeting the real-time requirement of each local node. We implemented the proposed traffic smoother on both the Linux and the Windows NT platforms, and conducted extensive experimental studies for various traffic-arrival patterns on the two testbeds. The studies showed that the message deadline-miss ratio can be kept well under 10^{-3} for *any* non-real-time traffic arrival rate if all the local nodes are equipped with the proposed traffic smoothers. Moreover, the studies showed that the proposed traffic smoother can provide a reasonable average throughput to non-real-time traffic while still yielding a remarkably low real-time message deadline-miss ratio.

We considered only the soft real-time communication part of a real-time control system, but our traffic smoothing can be extended to various other applications. In particular, we would like to explore ways of transporting real-time video over an Ethernet LAN using the proposed traffic smoother. Our traffic smoothing approach can also be applied directly to Fast Ethernet which is expected to replace 10 Mbps shared Ethernet for multimedia support.

References

[1] S.-K. Kweon, K. G. Shin, and Q. Zheng, "Statistical real-time communication over ethernet for manufacturing automation systems," in *IEEE Real-Time Technology and Applications Symposium*, June 1999.

[2] Y. Shimokawa and Y. Shiobara, "Real-time Ethernet for industrial applications," in *Proc. of IECON*, pp. 829–834, 1985.

[3] D. W. Pritty, J. R. Malone, S. K. Banerjee, and N. L. Lawrie, "A real-time upgrade for Ethernet based factory networking," in *Proc. of IECON*, pp. 1631–1637, 1995.

[4] R. Court, "Real-time Ethernet," *Computer Communications*, vol. 15, pp. 198–201, Apr. 1992.

[5] C. Venkatramani and T. Chiueh, "Supporting real-time traffic on Ethernet," in *Proc. of Real-Time Systems Symposium*, pp. 282–286, Dec. 1994.

[6] M. Molle, "100Base-T/IEEE802.12/Packet Switching," *IEEE Communication Magazine*, pp. 64–73, Aug. 1996.

[7] K. K. Ramakrishnan and H. Yang, "The Ethernet capture effect: analysis and solution," in *Proc. of 19th Conference on Local Computer Networks*, pp. 228–240, 1994.

[8] R. L. Cruz, "A calculus for network delay, part I: network elements in isolation," *IEEE Trans. on Information Theory*, vol. 37, pp. 114–131, Jan. 1991.

[9] B. Whetten, S. Steinberg, and D. Ferrari, "The packet starvation effect in CSMA/CD LANs and a solution," in *Proc. of 19th Conference on Local Computer Networks*, pp. 206–217, 1994.

[10] D. E. Comer, *Internetworking with TCP/IP Volume I, Principles, Protocols, and Architecture*. Englewood Cliffs, New Jersey: Prentice-Hall International, third ed., 1995.

[11] Microsoft Corporation, *MSDN library (CD-ROM)*. 1999.