A CONTINUOUS MEDIA COMMUNICATION SERVICE AND ITS IMPLEMENTATION [1,2]

Domenico Ferrari, Amit Gupta, Mark Moran, Bernd Wolfinger [3]
Computer Science Division
Department of EECS
University of California
Berkeley, CA 94270
and
International Computer Science Institute
1947 Center Street
Berkeley, CA 94704, USA.
{ferrari,gupta,moran}@tenet.berkeley.edu
wolfinger@rz.informatik.uni-hamburg.dbp.de

Abstract

Continuous media (CM) traffic often requires more network throughput and more stringent delay bounds than conventional discrete media traffic. At the same time, transmission requirements of CM traffic can usually be estimated more accurately, both in terms of the resources required for transmission and the time at which the next transmission request will occur. A data transport service has been designed that uses these considerations to provide a richer and more efficient service to CM traffic than message-based transport services. This paper briefly describes the service and a prototype implementation. The service is then evaluated via simulation experiments, which indicate that this service can often utilize network resources more efficiently for the transmission of CM traffic than message-based services.

[1]This research was supported by the National Science Foundation and the Defense Advanced Research Projects Agency (DARPA) under Cooperative Agreement NCR-8919038 with the Corporation for National Research Initiatives, by AT&T Bell Laboratories, Hitachi, Ltd., the University of California under a MICRO grant, and the International Computer Science Institute. The views and conclusions contained in this document are those of the authors, and should not be interpreted as representing official policies, either expressed or implied, of the U.S. Government or any of the sponsoring organizations.

[2]Presented at Globecom '92, Orlando Florida

[3]Current address: University of Hamburg, Computer Science Dept., Vogt-Koelln-Str. 30, D-2000 Hamburg 54

1 Introduction

The proliferation of computer workstations with fast CPUs, high-quality bit-mapped displays and support for audio I/O, along with the development of high-speed networks, has brought *continuous media*[4] into computer workstations and networks. Because CM traffic is expected to comprise a significant portion of future network traffic,[5] a separate data transport service designed for CM traffic could be justified *if* that service could transmit CM data more efficiently than a message-based service and/or if it could provide additional functionality to support CM clients.

A transport service designed for CM traffic would be able use a different traffic model that could allow CM clients to more naturally and more accurately characterize the rate and timing of their data transmission. This service could then use this information to more accurately estimate the network resources required to meet the client's performance needs. In addition, such a service could modify a client's traffic pattern (e.g., smooth out bursts) to further reduce resource requirements. A dedicated CM transport service can also reduce the utilization of end-system resources by using the regularity of CM traffic to predict future data transfer requests and thereby reduce the number of synchronous interactions between the client and the

[4]For this paper, we define *continuous media (CM)* to mean digital data that is generated/consumed isochronously at some granularity (e.g. motion video displayed at 30 frames per second), as opposed to discrete media data, which does not exhibit this regularity in time (e.g. file transfer, remote login).

[5]For instance, "live" transmission of uncompressed data from a standard color television picture would require approximately 24 Mbps.

service.

In addition to the efficiency advantages mentioned above, a dedicated CM data transport service can offer capabilities designed specifically for CM traffic. These additional capabilities could include abstractions for CM-specific error handling and for maintaining proper timing relationships within and between CM streams.[6] In addition, the elimination of synchronous interactions between the client and the service may simplify data transmission for many classes of clients. For example, data from a CM file-server, e.g. [And90] or a hardware compression module could be transferred directly to the transport service without an intervening process. At the receiving side, the CM transport service can write data directly into a buffer shared with a multimedia display server that is managing input and output devices and controlling the timing within and between streams (e.g. [AnH91]).

Section 2 of this paper describes a CM-specific transport service that provides the functional and efficiency advantages described above. Section 3 describes a prototype implementation of this service. Section 4 presents simulation experiments that show efficiency gains in the network for this transport service. For a more detailed description of the service and justification of the need for a dedicated CM transport service, the reader should see [WoM91] and [MoW92].

2 Description of service and protocol

We describe the *Continuous Media Transport Service* by first describing the interfaces between the service and the sending (C_S) and receiving (C_R) clients. We will then sketch the actions performed by service entities and the *Continuous Media Transport Protocol (CMTP)*[7] used for communication between them.

The service paradigm consists of two phases. During the setup phase, the sending client C_S requests a CM connection from the CMTP, specifying its traffic characteristics and quality of service (QoS) requirements. Two classes of service are supported: message stream and byte stream. CM traffic is specified as variable-size *frames* transmitted over fixed-length intervals. The *period* of the stream is defined as the duration of these intervals. For message streams, a frame is typically one message. For byte streams, a frame consists of all bytes generated for transmission during a period. The size of a frame is specified by its maximum and its mean, calculated over some averaging interval. A pseudo-minimum message size may also be specified to provide an additional characterization of burstiness. QoS requirements include bounds on end-to-end delay and data loss, and limited techniques for recovery from data loss. If CMTP is willing to handle the connection, it passes the request to the receiving client (C_R). In accepting the request, C_R is "promising" to consume data at the specified rate and accepting the risk of losing data otherwise. If C_R accepts the request, then the connection is established. In either case, C_S is informed of C_R's decision. Figure 1 depicts the essential interactions between the various components.

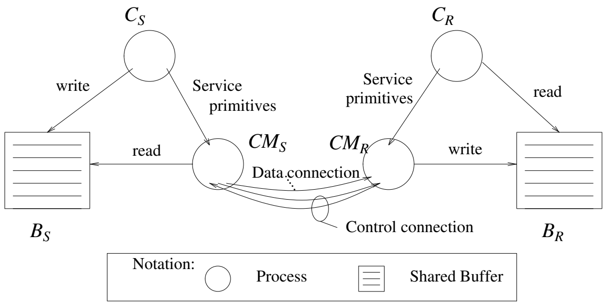

Figure 1: Basic components of CMTP implementation and illustration of their interactions

Data transmission on the CM connection occurs within *logical streams.* The stream mechanism helps provide synchronization between sending and receiving clients, and between connections. To transmit data on the connection, C_S first informs the service of the beginning of transmission by *opening a stream.*[8] After opening the stream, C_S simply writes data into a buffer it shares with the service (B_S). Once per period, the service then copies the data from the buffer it shares with C_S into the buffer it shares with C_R (B_R), satisfying the performance requirements specified during the setup phase. Since the timing of the stream is known, no synchronous interactions are required between the service and the clients.

CMTP is part of the real-time protocol suite being developed by the Tenet group at the University of California, Berkeley and the International Computer Science Institute. Setup requests are handled on behalf of CMTP by the *Real-Time Channel Administration Protocol (RCAP)* [BaM91]. RCAP reserves resources on both the sending and receiving end-systems, and establishes a network-level real-time connection via the *Real-Time Internet Protocol (RTIP)*, the network-layer protocol of the Tenet suite [ZVF92]. When the CMTP entity at the sending host (CM_S) is informed of the beginning of a stream, it likewise informs the CMTP entity at the receiving host (CM_R) via an *Open PDU*[9] and prepares for data transmission. Upon receiving an *Open*

[6] This capability is often referred to as *orchestration*.

[7] We use the acronym *CMTP* to refer to both the service and its associated protocol in the same way that *TCP* is used to refer to the DARPA transport service *TCP* and its associated protocol, the *Transmission Control Protocol*.

[8] Each connection handles at most one stream at a time.

[9] We are using the acronym *PDU* to refer to a Protocol Data Unit, following CCITT and OSI terminology.

PDU, CM_R informs C_R. Once per period, CM_S looks for data in the shared buffer and transmits it via RTIP to CM_R as one or more *Data PDUs.* CM_S must adjust its transmission rate so as to meet the performance guarantees promised to C_S without violating the traffic characterization specified to RTIP. Each *Data PDU* received by CM_R is written directly into the buffer it shares with C_R. Upon being informed of the end of a stream by C_S, CM_S empties B_S and then informs CM_R by sending a *Close PDU.* CM_R then informs C_R of stream closure. A more complete description of the service and protocol can be found in [MoW92].

3 Implementation Issues

The implementation of the CMTP service had to solve the following problems:

1. Choose a process structure for the CM transport service;
2. Guarantee mutual exclusion on access to shared data structures;
3. Enable the consumer of data (e.g. CM_S on thte sending side) to locate data;
4. Transfer data between producer and consumer;
5. Control and initiate data transfers[10]
6. Determine the size of the shared buffers;
7. Ensure compliance with the traffic characterization specified during setup.

To solve problem 1 in a UNIX[11] environment, we created a single CMTP daemon per end-system. When a new CMTP connection is established, the daemon at the sending host forks a child process to handle that connection. The child process then sets up the buffer to be shared with the client application. This choice greatly simplifies problem 2. The main shared data structure is the *Table of Open Connections*, which stores information about all CM connections. Mutual exclusion on this data structure is ensured by allowing each CMTP process to access only the entry associated with its connection, and by making the action of associating an entry with a process an atomic operation. An alternative process structure, consisting of a single multi-threaded CMTP daemon process was considered and rejected.

We solved problems 3 and 4 by using a buffer shared between the client and the service in both the sending and the receiving end-systems, thereby eliminating the need for most synchronous client/system interactions during data transmission. Use of this buffer also reduces the number of data copy operations required.

The implementation of the shared buffers introduced several additional problems. Because of the real-time requirements of our service, we chose to *lock* the shared buffers in physical main memory. Our implementation uses the appropriate UNIX system calls to perform this locking, which requires the CMTP server process to run as *suid root*, raising protection and security issues. The locking requirement also implies that CMTP may reject some connection establishment requests due to a lack of sufficient free physical memory. This part of the implementation is quite delicate, since locking excessive physical memory can deadlock the machine.

Because the shared buffer eliminated the need for a synchronous system call, we had to introduce additional mechanisms to allow the server to determine when to transmit data, and to provide producer/consumer synchronization between the client and the server (problem 5). To solve the control and initiation problem, the child process checks for data at the beginning of a stream, and then once per period after that. A naive CMTP implementation might use the following loop for scheduling data transfer:

```
while connection exists
 do
   pick up data from the buffer
   transmit the data
   sleep for 1 period
```

However, the above loop can *drift*, resulting in missed deadlines and continuing excessive delays in all subsequent data transfers. We have avoided this problem by implementing our code with a compensated waiting period.

Shared synchronization variables are used to provide producer/consumer synchronization for access to the shared buffer. These variables indicate the amount of data in the buffer and the amount of buffer that is still free. Each variable is written by one entity and read by the other. Descriptors are used when the client wishes message boundaries to be maintained by the service. Figure 2 depicts the use of these descriptors for the buffer shared between CM_R and C_R.

Finally, the size of the shared buffer must be determined (problem 6). The buffer requirements are a direct consequence of the fact that enough buffer space must be provided to absorb the maximum possible delay jitter and the maximum variations in the data rates observed by the sending and the receiving end-systems. The buffer requirements include *slack* to allow either the sending or receiving client to fall slightly behind. Each process checks the shared synchronization variables before modifying any

[10] Problems 3-5 have been addressed in [GoA91].

[11] UNIX is a trademark of USL

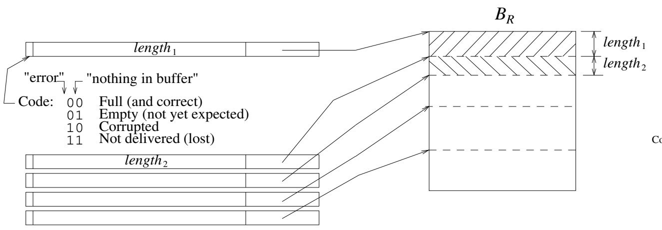

Figure 2: Shared buffer at the CM_R/C_R-interface with associated descriptors

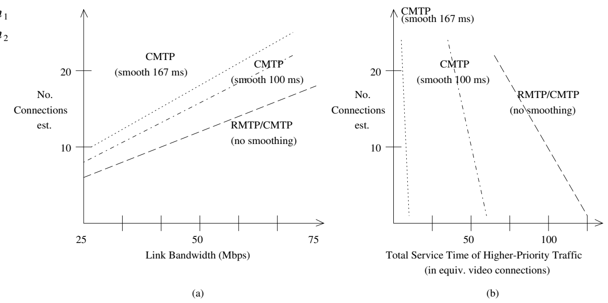

Figure 3: Number of connections established for each service, limited by (a) link-bandwidth, and (b) scheduling priority

part of the buffer. The size of the buffers needed has been calculated in [WoM91] and [MoW92].

Another implementation issue is ensuring that the CM service sends data at the appropriate rate (problem 7): if data is sent too rapidly, CMTP will violate the traffic characterization it specified to the underlying service (RTIP, in our case); if the data sending rate is too low, the client guarantees will be violated. Our implementation uses a credit-based scheme for ensuring that the service never violates either constraint. This scheme is described in [MoW92].

The CMTP service has been implemented and tested as a user process. Work to move the service into the UNIX kernel is nearing completion. Parallel work in the Tenet Group includes development of a real-time UNIX kernel. The CMTP implementation will use this kernel to ensure that the CMTP process gets correctly scheduled, as described above.

4 Experiments

Simulation experiments have been performed to verify that performance guarantees are met and to compare the number of connections that can be supported when using CMTP versus the number that can be supported when using the *Real-Time Message Protocol (RMTP)*, a transport protocol in the Tenet suite designed for discrete media [VeZ91]. For these experiments, we have assumed a point-to-point network with store-and-forward nodes. Video frames (320x240x8-bits grey scale) were compressed by first taking the difference from the previous frame and then applying the quadtrees algorithm [TaP89] to the result; however, every 15th frame was compressed with the quadtrees algorithm alone to allow for recovery from data loss. The resulting stream had a 5:2 ratio between the maximum and mean frame sizes, and required approximately 800 kbps. We assumed that an end-to-end delay of 300 ms was acceptable.

Two classes of experiments were conducted, in which the limiting factor for connection establishment was, respectively, link bandwidth and scheduling priority in the nodes. Because CMTP allows a sender to more accurately express the burstiness of its traffic than RMTP, CMTP can often smooth the traffic pattern offered to the network, thereby lowering the maximum data rate required without violating the QoS requirements. Since RMTP could not model the burstiness of the stream as accurately, it often had to reserve more bandwidth to provide the same guarantees.

In the bandwidth-limited experiments, the number of connections that could be supported was linear in the link bandwidth. In these experiments, RMTP and CMTP actually used approximately 30% and 50% of the bandwidth reserved, respectively, with the result that CMTP was able to support approximately 67% more connections for identical networks and workloads (see Figure 3a).

Since smoothing requires the introduction of delay, any smoothing of the traffic pattern will affect the number of connections that can be supported when scheduling priority is the limiting factor. Therefore, in the experiments in which scheduling priority is the limiting factor, RMTP was able to support significantly more connections than CMTP, *when CMTP chose to smooth the traffic*. Of course, CMTP can choose not to smooth the traffic, in which case it can support the same number of connections as RMTP (see Figure 3b). We should note that, in order to force scheduling priority to become the limiting factor in our network, we had to introduce an additional connection, which had a low bandwidth requirement but a higher scheduling priority than the video connections and a (non-preemptive) service time equal to that of over 120 video connections (i.e. a situation equivalent to the existence of over 120 pre-established, "low-bandwidth" video connections). Because of the high data rates and relatively loose delay bounds of the video connections we established, we expect it to be much more likely that connection establishment will be limited by link bandwidth availability rather than by scheduling priority. Therefore, we expect that the traffic smoothing option enabled by the more accurate traffic characterization of CMTP will significantly increase the number of video connections that can be supported in a network.

Simulation experiments conducted by Eckhardt Holz [Hol92] have studied the boundary conditions under which CMTP is able to meet performance guarantees made to C_S. His simulations showed that if the sending CMTP process fails to receive adequate processing time, data will be lost at the sender due to buffer overflows.[12] Holz also verified the claim made in [MoW91] that starvation and/or data loss caused by variability in network delays can be avoided by pre-filling the shared buffer at the receiver (B_R) before the receiving client (C_R) begins consuming data.

5 Summary and future work

This paper has briefly described CMTP, a data transport service for continuous media traffic. The key contributions of CMTP include: a new paradigm for data transport geared toward CM clients, and a traffic model and service that can utilize *a priori* knowledge about the timing of future data transmission to more efficiently utilize network and end-system resources. The implementation of a prototype has been described, along with simulation experiments verifying the expected efficiency gains.

In the near future we will experiment with the CMTP prototype in both a local-area, and two wide-area, high-speed testbeds. In these experiments we hope to verify the timing and correctness of the protocol for a variety of client/network mixes; compare the number of connections supported by RMTP versus CMTP; and measure the buffer utilization, with the aim of tightening up the calculations for buffer requirements.

Some key questions still remain. The stream delays resulting from our shared-buffer approach may be too high for some interactive applications, particularly when prefilling the receiver's buffer to compensate for jitter in data rate and network delays. Also, the buffers required can be quite large for some applications, e.g., large video conferences. For such applications, we may need to relax our current requirement that the buffers be *locked* in physical memory.

6 Acknowledgments

The authors would like to express their particular gratitude to Francesco Maiorana (NYU) for his participation in the implementation of the prototype and to Eckhardt Holz for his detailed simulation study to analyze the behavior of the CMTP service under various boundary conditions. In addition Riccardo Gusella and Ramesh Govindan have provided valuable information and feedback during the design and implementation. We would like to thank the Tenet group at ICSI and U.C. Berkeley for their input during a large number of in-depth discussions, especially Hui Zhang, Tom Fisher and Dinesh Verma.

[12] Holz assumed that the sending client, C_S, continued to generate data at the specified rate.

References

[And90] D. Anderson, "Meta-Scheduling for Distributed Continuous Media", UC Berkeley, EECS Dept., Technical Report No. UCB/CSD 90/599, (October 1990).

[AnH91] D.P. Anderson and G. Homsey, "A Continuous Media I/O Server and Its Synchronization Mechanism", IEEE Computer, Vol. 24, No. 10, October 1991, pp. 51-57.

[BaM91] A. Banerjea and B. Mah, "The Real-Time Channel Administration Protocol", Proc. 2nd Int. Workshop on Network and Operating System Support for Digital Audio and Video, Heidelberg (November, 1991).

[GoA91] R. Govindan and D. Anderson, "Scheduling and IPC Mechanisms for Continuous Media,“ Proc. of Symp. on Operating System Principles, pp.68-80, October 1991.

[Hol92] E. Holz, personal communication, August, 1992.

[MoW92] M. Moran and B. Wolfinger, "Design of a Continuous Media Data Transport Service and Protocol", International Computer Science Inst. TR-92-019, (March 1992).

[TaP89] S. L. Tanimoto and Pavlidis, "A Hierarchical Data Structure for Picture Processing," Computer Graphics and Image Processing, no. 4, pp. 104-119, 1989.

[WoM91] B. Wolfinger and M. Moran, "A Continuous Media Data Transport Service and Protocol for Real-Time Communication in High Speed Networks", Proc. 2nd Int. Workshop on Network and Operating System Support for Digital Audio and Video, Heidelberg (November, 1991).

[VeZ91] D. Verma and H. Zhang, "Design Documents for RTIP/RMTP", unpublished (1991).

[ZVF92] H. Zhang, D. Verma and D. Ferrari, "Design and Implementation of the Real-time Internet Protocol", IEEE Workshop on the Arch. and Impl. of High Performance Communication Subsystems, Tucson AZ, (Feb 1992).