A Continuous Media Data Transport Service and Protocol for Real-Time Communication in High Speed Networks *

Bernd Wolfinger [1]
Mark Moran

The Tenet Group
Computer Science Division, Department of EECS,
University of California, Berkeley
and
International Computer Science Institute
Berkeley, CA 94720, USA.

ABSTRACT

An important class of applications with real-time data transport requirements is defined by applications requiring transmission of data units at regular intervals. These applications, which we call continuous media (CM) clients, include video conferencing, voice communication, and high-quality digital sound. The design of a data transport service for CM clients and its underlying protocol (within the XUNET II project) is presented in this paper. The service makes use, in particular, of an a priori characterization of future data transmission requests by CM clients.

First, we will give a few examples of CM clients and their specific data transmission needs. From these clients, we then extract a generalized list of data transport requirements for CM and describe the basic features of a service designed to meet these requirements. This service provides unreliable, in-sequence transfer (simplex, periodic) of so-called stream data units (STDUs) between a sending and a receiving client, with performance guarantees on loss, delay, and throughput. An important feature of the solution is the use of shared buffers to eliminate most direct client/service interactions and to smooth traffic patterns, which may be bursty due to fluctuations in the arrival process of data and variability of network delays. The paper concludes with some aspects of implementation.

1. Introduction

Applications with real-time data transport requirements fall into two categories: those which require transmission of data units at regular intervals, hereafter referred to as *continuous media (CM) clients*, and those which generate data for transmission at relatively arbitrary times, hereafter referred to as *(real-time) message-oriented clients*. Examples of the former are video conferencing, in which video frames (of fixed or variable length) are sent from source to destination once per frame time (e.g. 33 ms), voice communication, playback of high-quality digital sound, and transmission of sensor data that is measured and transferred with strict periodicity. Examples of real-time message-oriented clients are those which require urgent messages or transactions, and mail service with guaranteed delivery latency.

It is generally accepted that dedicated transport protocols are necessary for high speed networks. Adaptation of existing transport protocols, originally designed for lower speed networks (such as OSI Transport Protocol Class 4 or TCP), to high speed environments is not straight-forward, and may not provide satisfactory performance to transport

---

* 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.

[1] On sabbatical leave from the University of Hamburg, Computer Science Dept., Bodenstedtstr.16, D-2000 Hamburg 50

service users of future networks. Therefore, considerable research has been conducted in the design of completely new transport protocols to support high speed end-to-end communication between users. Surveys of the general requirements these protocols must satisfy, and of existing protocol proposals can be found in e.g., [DDK90], [LPS91], [WrT90], [Zit91]. In these publications, it is suggested that new algorithms be developed to support basic transport protocol functionality (such as flow control, error detection and correction, connection management, etc.). In addition, the use of specific implementation techniques, e.g. parallel processing, is advocated. Current transport protocols designed for high speed networks include, e.g., Delta-t Transport Protocol, cf. [Wat89], Network Block Transfer Protocol (NETBLT), cf. [CLZ87], Versatile Message Transaction Protocol (VMTP), cf. [ChW89], Express Transport Protocol (XTP/PE), cf. [Che88], and the protocol designed by Netravali et al. and described in [NRS90].

The literature suggests general agreement among network designers that transport protocols should be tailored to meet the various transport service requirements of end users. Requirements of various users can be supported by using rather general transport services and providing options to flexibly adapt the service to the differing requirements of users (e.g. during establishment of a transport connection). Alternatively, it is also possible to split the transport service *a priori* into two (or more) different, cleanly separated, services. Each service would support a class of users with similar data transport requirements. This second solution is chosen in this paper, as we will describe a transport service designed for continuous media clients, which we expect to coexist with a transport service designed for message-oriented clients. For surveys on the requirements of continuous media applications for data transport, the reader is referred to [HSS90] and to [ITC91]. Requirements of video transfer, in particular, can be found in [LiH91]. Section 2 details the arguments in favor of a separate transport service for CM clients.

2. Necessity of a Dedicated Transport Service for Continuous Media Applications

Because CM clients are better able to characterize their future behavior than message-oriented clients, a dedicated CM data transport service can potentially provide them with a more cost-effective service by characterizing the future resource demands of such clients a priori.

In addition to the efficiency advantages mentioned above, a dedicated CM data transport service could provide better functionality for CM clients than a traditional message-oriented transport service, because of the many incompatible functional requirements of these two classes of applications. For example, a dedicated transport service could provide the abstraction of *logical (data) streams* between CM clients. A *stream* here denotes a continuous sequence of data units (continuous only with respect to some limited granularity in time) provided and to be transmitted during a given time interval, which corresponds to the duration of the stream. As we also assume in-sequence delivery of the data units of a stream, we can consider such streams to be generalization of Lixia Zhang’s "flows" (defined as a "stream of packets that traverse the same route from the source to the destination and that require the same grade of transmission service" in [Zha91]). At any instant in time, each established connection between two CM clients is used by at most one stream. By making such streams visible to the data transport service, network and system resources could be conserved between streams. More importantly, some connection parameters may be redefined for the duration of the stream, allowing conservation of resources and providing better cooperation between sender, receiver, and the transport service (e.g. the current stream could be stopped and a new, slower stream started to support video playback with "freeze frame" and "slow motion"). Streams also provide a natural mechanism for synchronizing data from different connections (e.g. video and audio). Such streams would be difficult to implement on top of a message-oriented transport service, because of the requirement that streams (and their associated parameters) be visible to the transport service. Since this functionality would not be used by most message-oriented clients, implementing a new service is preferable to adding this functionality to a message-oriented service.

A dedicated CM transport service can also provide error-handling mechanisms that are more suitable for CM clients. Although most message-oriented clients cannot use data which is only partially correct, many CM clients can tolerate limited data loss, and some can even utilize corrupted data. [HTH89]

Finally, while message-oriented clients cannot predict the time of their next data transfer, and so must explicitly inform the system to initiate each transfer, the time of each data transfer for a CM client can be deduced; therefore, the requirement that data transfer be initiated via an explicit interaction with the system introduces unnecessary overhead. This situation is exacerbated when data is provided for network transfer by a DMA-like device, e.g. a hardware coder-decoder (codec) for compressed video, since a (user) process must intervene between data generation and transmission.

The above differences in data transport requirements between message-oriented and CM clients justify design of a dedicated CM data transfer service which could provide better service for CM applications in four ways:

(1) a better traffic model for characterizing CM traffic and specifying performance requirements;

(2) the abstraction of (logical) streams, which are visible to the transport service;

(3) CM specific error handling, including delivery of all correctly received data and (possibly) of corrupted data, and of optionally replacing corrupted/lost data with dummy data; and

(4) the elimination of the need for a rendezvous (e.g. via a system call) between the client and the service for each data transmission.

At this point, we would like to emphasize that there is no fundamental reason a message-oriented transport service could not offer a service that included (1) - (3); however, as we argued above, these capabilities are neither required nor desirable for most message-based applications, and hence it seems wiser to implement a new service to provide them to CM clients.

3. Data Transport Requirements of Continuous Media Clients

In this section, we will give a few examples of CM clients and their data transport requirements. From these clients we extract a generalized list of data transport requirements for CM. In section 4 we will describe the basic features of a service designed to meet these requirements. All the applications listed require a strict upper bound on delay and on delay-jitter, where delay-jitter is defined as the difference between the maximum and minimum delay. If an upper bound is provided on delay, delay-jitter will result only in early delivery, and can therefore be absorbed by buffering in the service provider if enough buffers are available. Therefore, delay-jitter is not listed as a requirement for the clients. However, it should be noted that a network which controls jitter would allow less buffer space to be allocated to the connection on the receiving end-system.

Typical continuous media applications include:

- *Compressed Video:* Variable rate; delays <= 300 ms, if interactive; DMA-like coding devices; loss tolerant; synchronization with audio or text; variable period (slow-motion or fast-forward) or suspend transmission (freeze frame).
- *Uncompressed CD quality digital audio:* Constant rate; loss sensitive.
- *Multiplexed, interactive digital voice:* Constant rate; delay <= 300 ms; loss tolerant; silent periods.
- *Multimedia distributed classroom:* Phases of instruction (e.g. lecture, movie); interactive.

From these examples we have determined the following list representing data transport requirements of *most* CM clients:

(R1) Periodic delivery of data without gaps.

(R2) Bounded delay of stream between sender and receiver.

(R3) Logical stream abstraction: used to communicate redefinition of stream parameters to receiving client and data transport service provider, and for synchronization of data from separate connections.

(R4) Delivery of all correctly received and possibly of corrupted data.

(R5) Notification of undelivered or corrupted data.

(R6) No requirement for explicit interactions with system for each data transfer (i.e. delivery of data is periodic, not event-driven).

4. Definition of the Continuous Media Transport Service

We now briefly describe a continuous media transport service (CMTS) designed to meet the needs of CM clients (a considerably more detailed specification of this service and its underlying protocol is given in [MoW91]). This service provides unreliable, in-sequence transfer (simplex, periodic) of *stream data units* (as defined below) between a sending client (C_S) and a receiving client (C_R), with performance guarantees on loss, delay, and throughput. The CMTS service satisfies all of the requirements (R1),...,(R6) as listed above. Data is passed from C_S to the CMTS entity on its end-system (CM_S) via a shared circular buffer. Synchronization between C_S and CM_S is provided via traffic and performance parameters, and through explicit synchronization variables.

All traffic and performance parameters are defined in relation to two basic units: the *stream data unit (STDU)* and the *periodicity* of the conversation (T). An STDU is a data unit whose boundaries must be preserved by the CMTS and indicated to C_R. It is C_S, which decides how the stream to be transferred to C_R is mapped onto a sequence of STDUs as illustrated in Fig. 1. Typically, for a CM application, the information to be transmitted (e.g. voice signal, sequence of images) is digitalized prior to its transmission by a coding process (or possibly a sequence of such processes). The coding process maps a continuous signal function (in the sense of coding theory, e.g. voice, moving scene) onto a sequence of code words (cf. coding theory again). A CM application then has several options in mapping these code words into a sequence of STDUs:

a) Sequence of code words mapped onto sequence of bytes (byte stream), 1 byte corresponding to 1 STDU;

b) one-to-one mapping of code words onto STDUs;

c) concatenation of several code words to build one STDU, e.g. in the case, where different sub-streams are multiplexed (time-multiplex) by an application to form one overall stream;

d) combination of a code word and a time-stamp into one STDU (where a "time-stamp" is a reference to the period the code word represents); e.g. relevant in transmitting a stream, which had been stored, along with the timing information required to reconstruct its original timing.

The periodicity, T, of a stream can also be specified by an application. The periodicity characterizes the frequency of coding events (typical values chosen for periods would be $T=k\times0.125$ ms in PCM-voice coding or $T=k\times33$ ms in transfer of a video stream, where k is an integer). The data corresponding to a period maps into an integral number of STDUs. The CMTS service recreates on the receiver, the stream that had been seen on the sender at the granularity (in time) of a period, T. This implies, in particular, that data associated with period Δt_i at the sender must arrive before the beginning of the corresponding period $\Delta\tau_i$ at the receiver. Of course $|\Delta t_i|=|\Delta\tau_i|=T$, for all i, if $|I|$ denotes the length of interval I. Fig. 2 illustrates the basic timing within transfer of a sample stream (e.g. of voice or video data). It should be noted, however, that this figure does not reflect the fact, that we allow some "work-ahead" to the sender of the stream and some time "behind-schedule" to the receiver, as described below.

The conversation between C_S and C_R consists of a sequence of logical streams, with intervals of (medium- or long-term) silence between streams. A logical stream consists of a periodic sequence of STDUs which may be fragmented for actual transmission by the network. C_S must inform CM_S of the beginning and end of each logical stream. The model we are using for interactions between C_S and C_R is that data generated during an interval Δt_i by C_S is needed by C_R during the corresponding interval $\Delta\tau_i$, where $t - t_0$ represents the elapsed time of the stream at the sender and $\tau - \tau_0$ represents the elapsed time of the stream at the receiver (t_0 and τ_0 denoting beginning of stream as observed by sender and receiver respectively). Therefore, after indicating the beginning of a logical stream, C_S is *obligated* to ensure that all the STDUs corresponding to a given period are in the shared buffer before the end of that period, where periods are defined at regular intervals (of length T) from the beginning of stream transmission (t_0 in this case). Because a sender may have difficulty providing data corresponding to an interval exactly within that interval (e.g. due to contention for CPU or memory bus), another parameter, N_{Sslack}, is defined as the number of bytes, C_S is allowed to provide to CM_S ahead of schedule (i.e. before t_{i-1} for Δt_i). The solution of prefilling buffers has also been suggested in [AHS90]. C_S is also obligated to obey the traffic characteristics specified for the conversation and for the current stream.

At the beginning of a logical stream, C_S may redefine some traffic and performance parameters for the duration of the stream. These must be no more strict than the parameters of the actual connection (e.g. a decrease in the data rate is allowed). Such information may be used to assist cooperation between C_S and C_R because the characterization of a stream can more accurately represent current traffic and performance needs than the long-lasting connection parameters, which must cover all possible streams to be sent on the connection. The stream characterization may also allow system and network resources to be conserved. In addition, C_S may indicate a value for $d_{Sstartup}$, the time between the time of the request, t_{start}, and the start of the stream, t_0, at the sender. This time is used by C_S to pre-load the buffer. An upper bound on the *duration* of the stream may also be specified, to be used in allocating resources at the receiver.

After being informed of the start of a new logical stream, CM_S is obligated to cooperate with CM_R to transfer all STDUs for each period to C_R before the beginning of the corresponding period on the receiver, provided that C_S met its contractual obligations.

The CMTS entity at the receiving end-system (CM_R) and C_R also interact via a shared circular buffer, stream parameters, and shared synchronization variables. CM_R must inform C_R of the beginning of a new logical stream. After that, CM_R is obligated to put data in the shared buffer before the beginning of the period in which it will be needed (i.e. before time $\tau = \tau_{i-1}$ for period $\Delta\tau_i$), and C_R is obligated to remove from the shared buffer all the STDUs corresponding to a period before the end of that period (i.e. before time $\tau = \tau_i$ for period $\Delta\tau_i$). Since a receiver may have difficulty removing data exactly during its corresponding period, the value N_{Rslack} is defined at the receiver to indicate how far C_R can fall behind without data being lost. More precisely, C_R can leave up to N_{Rslack} bytes in the buffer after the end of their corresponding period. As at the sender, the first period is defined to begin after a delay of length $d_{Rstartup}$, where $d_{Rstartup}$ includes $d_{Sstartup}$, as well as the delays introduced for smoothing and for tolerating delay-jitter in the network. In addition, CM_R must inform C_R of data loss (which includes late data) and corrupted data. In the implementation, STDU descriptors are used to maintain STDU boundaries and to indicate errors in the data.

In order to meet guarantees regarding buffer overflow and starvation avoidance at the receiving client, all four entities must be involved in a handshake at the beginning of the conversation so that each may approve traffic and performance parameters. At this time some of the entities may reserve system and network resources to ensure that they will be able to fulfill their contractual obligations. This handshake is accomplished as follows: C_S presents a proposed set of parameters to CM_S. If CM_S accepts these, it passes them on to CM_R, with some possible modifications and additions. If CM_R accepts the (revised) parameters, it passes a (possibly) revised version of the original set to C_R. If C_R accepts the conversation request, it informs CM_R, who informs CM_S, who informs C_S. The parameters of the handshake before a conversation are described below for the C_S/CM_S interface. Parameters for the handshake between other entities are analogous to those described here.

The parameters of the C_S/CM_S interface were chosen to capture those traffic characteristics of continuous media traffic that have the greatest impact upon resource requirements and to specify performance requirements applicable to CM applications. The most significant of these parameters are described below. (A full description can be found in [MoW91].)

Traffic characterization parameters

The first three traffic parameters allow for flexible definition of a CM connection. They cannot be changed for individual streams. $STDU_{\max}$ specifies the maximum size of an STDU (in bytes), i.e. maximum size of a logical unit for which boundaries must be maintained. ($STDU_{\max} = 1$ is allowed as a special case, leading to a transparent byte stream data transfer). *CONST_SIZE* and *CONST_NUM* are booleans used to further describe the type of service requested (i.e. byte stream, constant-size STDUs or variable-size STDUs).

The next group of parameters characterize the traffic pattern. All traffic and performance parameters are defined in terms of T, the basic period of the stream. $S_{\max}$ is the maximum amount of data (in bytes) which C_S may put in the shared buffer during *any* period. S_{avg} specifies an upper limit on the mean number of bytes presented by C_S for transport within a single period, calculated over any averaging interval consisting of I_{avg} consecutive periods. $S_{\min}$ provides a limit on the burstiness of the stream missing from other traffic specifications we have seen. Our model of CM streams is that even variable-rate streams will have *something* to transmit each period. $S_{\min}$ allows a user to specify the minimum amount of data which is *expected* to be transmitted each period. One common example of such a stream is a compressed video stream which transmits an image compressed with only intra-frame encoding followed by several frames that achieve higher compression using inter-frame encoding ([e.g. Leg91]). Without this parameter, we would have to make the worst-case assumption that all the data allowed in an averaging interval could be sent in a minimal number of periods with no data being transmitted during the rest of the averaging interval. (Note: C_S is *not required* to transmit $S_{\min}$ bytes each period, but if it does not, it cannot be sure of sending its entire allotment for the averaging interval without possible loss of data due to buffer overflow.)

We would like to briefly discuss some of the advantages of the traffic characterization described above for CM clients. As far as the authors are aware, present transport services allow clients to describe their burstiness by specifying a variability in the inter-arrival times of fixed-size messages (e.g. [AHS90], [FeV90]). While this is the proper characterization for network packets and for message-oriented clients, it is not a convenient manner for describing the burstiness of CM traffic (e.g. compressed video), which is better described as a variable-size message sent at fixed intervals. The characterization we have presented allows for variable-size "messages" (corresponding to the data produced in a period) at a fixed interval T. This characterization, along with the inclusion of $S_{\min}$ will allow for a better characterization of CM traffic and, therefore, more efficient utilization of network and end-system resources.

N_{Sslack} is the maximum workahead allowed to C_S as previously mentioned. It specifies the maximum number of bytes which C_S is allowed to put in the shared buffer early, i.e. before the beginning of the period to which the data corresponds.

Quality of service parameters

The parameters for indicating the quality of service (QOS) desired were chosen to sufficiently communicate the needs of most CM clients. As stated previously, the basic model of the CMTS service is that the stream on the sending end-system will be recreated on the receiving end-system at the granularity of a period. The service handles delay-jitter for the client (where delay-jitter is defined as the variability in delay) by ensuring the shared buffer at the receiving end-system is large enough to tolerate maximum possible jitter without overflow. Because there are no explicit interactions between the client and the service, the implications of delay-jitter on timing of the stream do not apply.

D_{stream} is the maximum acceptable delay of the stream, where the delay of a stream is defined as the time between the start of a stream at the C_S/CM_S interface and the start of the stream at the C_R/CM_R interface. Since D_{stream} must also be maintained for each period of the stream, it implies a deadline for data arrival at the receiver. (All data associated with the *i*th interval on the sender, Δt_i, must arrive at the receiver before the beginning of the same period at the receiver, i.e. before τ_{i-1}.) S_{err} allows the client to specify the maximum *granularity* (in bytes) of a data loss caused by data corruption or buffer overflow. This parameter is interpreted as an upper bound on the packet size used for this stream. W_{err} specifies a lower bound on the probability that a unit of data transfer (of size $\leq S_{err}$) is correctly delivered to the receiving interface. *REPLACE* is a boolean that indicates whether corrupted data should be replaced with dummy data (supplied by the user in a dummy data unit of size S_{err}) instead of being delivered as it is received or discarded. This service is useful for in-band signalling of data loss and for filling in holes in the data stream.

5. Basic Underlying Services and a Transport Protocol to Support CMTS

We presently implement, within the XUNET II project, the service described above as part of a real-time protocol suite. Connection establishment and teardown (including resource allocation) are provided by a connection administration service (RCAP, cf. [BaM91]), which will also handle the connection establishment and teardown functions for CM connections. A network service (RTIP, cf.[VeZ91]) which implements the schemes described in [FeV90] will provide network connections with real-time guarantees for delay, delay-jitter, throughput, and loss. RTIP will allow the transmission of packets via connections established within (possibly a hierarchy of) interconnected subnetworks with FDDI- and ATM-components.

In addition to these underlying services, the CMTS service requires relatively large buffers on the receiving end-system, as one of the main concepts underlying this service is to use buffers to smooth fluctuations in the arrival process of data during a stream as well as delay jitter introduced by the network and/or the end-systems. (Calculations with respect to buffer requirements will be given in section 6). A real-time clock with a high precision timer, and real-time scheduling of the CPU and network driver are also required.

To realize the CMTS, we defined the *Continuous Media Transport Protocol (CMTP)* which supports communication between CMTS peers at the sender and the receiver. The first version of the CMTP protocol could be kept relatively simple. This results primarily from the fact that several of the communication functions needed in conventional data communication (in particular, retransmissions for error correction, flow control, etc.) are not required in order to provide the CMTS service. Regarding retransmissions, we take the position (stated in [FeV90]) that most real-time applications will not be able to wait for retransmissions, and even if they could, the amount of data which would need to be stored to perform retransmissions on a high bandwidth-delay product network could not be justified for CM clients, which do not require perfectly reliable service. Similarly, resetting a data stream to an earlier status (period) is not possible as the resource requirements needed to set check-points in general are prohibitive for storing an intermediate status of a stream. Therefore in the case of a serious error, tear-down of a connection with successive re-establishment of a connection and initialization of a new stream seem to be the most appropriate measures. The simplex nature of the real-time connections used also represents an obstacle to a dialog between sender and receiver directly within such a connection. So, our assumption in the design of the CMTP protocol has been that a data stream between C_S and C_R is transmitted via exactly one (uni-directional) connection between C_S and C_R. This connection thus represents a data connection. As no multiplexing takes place in the CM transport layer, a one-to-one mapping of the addresses of communicating clients to the address of the network connection (connection-id provided by RCAP in the case of XUNET II) can be used to solve the addressing problem.

As an extension to the current design, we assume that CM_S and CM_R are able to exchange (reliably) control information concerning the state of the data connections presently established between them. In a similar way, we assume that the CMTS service reliably transfers C_S- / C_R-control information between clients to support an application-oriented protocol between them (separate communication, in addition to the exchange of a data stream). This solution can be viewed as an "out-of-band-signaling" between the communicating CM clients.

Until now, the CMTP protocol has only been specified for covering communication within the data connection. Experiences of the CMTS implementation are considered to be indispensible prior to a protocol extension and will be taken into account in the completion of the CMTP protocol.

The first version of the CMTP protocol is based on three types of protocol data units (PDUs):

- ON_PDU: Signal start of stream; redefine parameters, facilitate synchronization
- OFF_PDU: Signal end of stream
- DATA_PDU: Transmit one STDU, one fragment of an STDU or a number of bytes (in case of a byte stream).

The beginning of a stream must first be signaled by the sending client via an ON_PDU. Reaction after receipt of an ON_PDU is according to the service specification (cf. section 4). To increase reliability, two copies of an ON_PDU are transmitted at the beginning of a stream (separated by an interval, which is considered to be large enough in order to make errors in both ON_PDU transmissions sufficiently independent of each other). The receipt of at least one of these copies by CM_R is sufficient for the correct initialization of the stream. Loss of both ON_PDUs will be considered as a serious error situation, requiring connection tear-down and re-establishment.

After stream initialization (if successful), DATA_PDUs can be sent to transfer data of the stream. Data of the stream (i.e. the STDUs) is mapped onto DATA_PDUs either 1:1, by concatenation of STDUs (in particular in byte streams) or by fragmentation of STDUs (respecting maximum packet-size of the underlying network service as well as maximum size of acceptable loss, specified by C_S). Each DATA_PDU is transported in exactly one packet.

The current stream ends when an OFF_PDU arrives, indicating a silent period will follow. An alternating 1-bit stream identifier is used to cleanly separate successive streams from each other, even in the case of error situations (e.g. loss of successive OFF and both ON_PDUs).

Additional control information exchanged between CM_S and CM_R (protocol extension) could refer, e.g., to

- acknowledgement (ACK/NAK) for ON_PDU;

- acknowledgement (ACK) for OFF_PDU;
- indication of buffer overflow at receiving interface to CM_S;
- PING to check whether sender is still alive, when neither data nor an OFF_PDU has been received for a given interval; and
- some control signal, used by the sender to check whether the receiver is still alive.

6. Some Implementation Considerations

Because of the desire to eliminate most client/system interactions, data is transferred between the service provider and clients via a shared circular buffer at both the sending and the receiving site (C_S/CM_S- and C_R/CM_R- interfaces). This implementation also lowers the number of data copy operations. Fig. 3 depicts some essential types of interactions between components used in the implementation.

Shared synchronization variables are used to inform the producers of data (C_S at the sender and CM_R at the receiver side) as well as the consumers (CM_S at sender and C_R at receiver) of the present state of the buffer (e.g. amount of data in the buffer). STDU descriptors are used to delineate (variable-size) STDUs and to provide an indication of data errors without explicit interactions between clients and the service provider. Fig. 4 depicts the use of descriptors for the buffer (B_R) shared between CM_R and C_R.

In the prototype implementation, CM_S will check the buffer (B_S) it shares with the sending client once per period, packetize the data found there (up to a maximum controlled by the maximum burst and average rates as required by the underlying network service), and schedule its transmission.

[MoW91] contains the derivation of the following conservative estimate for b_S, the minimum size of buffer B_S guaranteed to prevent data loss due to overflow of this buffer:

$$
\begin{aligned}
b_S = {} & 2 \times S_{\max} && \textit{(data for current and next periods)} \\
& + N_{Sslack} && \textit{(workahead for } C_S\textit{)} \\
& + b_{sm} && \textit{(for smoothing)} \\
& + b_{align} && \textit{(for aligning STDUs in buffer)}
\end{aligned}
$$

Similarly, for the size, b_R, of the buffer B_R, we obtain:

$$
\begin{aligned}
b_R = {} & b_s && \\
& + N_{Rslack} && \textit{(amount of data } C_R \textit{ can be late in consuming)} \\
& + 2 \times S_{\max} \times \left\lceil (d_j / T) \right\rceil && \textit{(delay jitter in ON\_PDU and stream transmission)}
\end{aligned}
$$

The buffer requirements are a direct result of the fact that enough buffer space must be provided to absorb the maximum possible delay jitter and variation in the data rate being observed at the sending and receiving end-systems. Buffer requirements at the receiver can be reduced if CM_S uses additional information with respect to the timing of the stream in order to delay the transfer of STDUs at the sending end-system, thereby absorbing workahead at the sending end-system. For a connection being used to transport fixed sized STDUs (in particular for a byte stream), b_align = 0, as alignment problems do not exist in these cases.

7. Summary

Most important properties of the transport service and protocol introduced in this paper are a consequence of our effort to tailor the service and protocol design to the data transport requirements of continuous media applications. Therefore, the service introduced supports the continuous delivery of a data stream to a receiving CM client, error-handling mechanisms that can be adapted flexibly to the typical demands of CM applications, the use of a priori knowledge with respect to the future arrival pattern to be expected for a given stream, etc. The basic concepts introduced, such as the notion of stream data unit, as well as the large variety of parameters offered at the service interface can be used by communicating CM applications for a relatively flexible characterization of (e.g. voice or video) streams. This flexibility is also provided for the mapping of STDUs onto packets of an underlying network service (1:1, fragmentation or concatenation as options), where the mapping may even be controlled by the transport service users (e.g. by specifying the maximum granularity of a data loss). The solution chosen can easily support the possibility of allowing a (de-) coding process to react to the state of the communication system, as suggested e.g. in [GiG91]. The communication system's state might be considered to be reflected by the actual occupancy of the shared buffers (B_S and B_R) on sending and receiving end-systems, which could lead to a variation of the (de-) coding rate.

To complete the present design it will still be necessary to integrate the experiences gained in the prototype implementation of CMTS in an extended service/protocol design. The extensions will have to specify, in particular, additional possibilities of reacting to protocol errors as well as the exchange of different types of control information.

Limitations of the solution primarily concern the buffer requirements in the end-systems, which may become significant in those cases when delay jitter within the network and within the end-systems will become large and additionally large traffic fluctuations exist within the arrival process of the stream. However we believe that in future computer systems (even in workstations and personal computers) we can expect provision of communication buffers in the range of (a few) MByte at least for CM applications, if this leads to significant simplifications and performance improvements. For some dialog-oriented applications the stream delay resulting from our approach (of typically > 100 ms) may become disadvantageous as well. Realization of multi-point connections (e.g. required in video-conferencing) by means of (a possibly large number of) point-to-point connections, which would be the solution based on the CMTS service (within XUNET II architecture), may also lead to some inefficiencies. These inefficiencies could, of course be eliminated if the underlying network service would support multipoint communication.

In parallel to the CMTS prototype implementation, presently a modeling study is being carried out in order to gain some insight into the impact that configuration parameters of end-systems (such as buffer sizes, run-times of communication software, etc), properties of the underlying network service (such as packet delay jitter, packet loss rate, etc.), and the local load of the end-systems may have on the quality of the CMTS service as observed by CM clients (e.g. expressed by the probability of a buffer overflow with resulting loss of data and/or by the probability of late arrival of data in the B_R buffer).

8. Acknowledgements

The authors would like to express their particular gratitude to Amit Gupta and Francesco Maiorana for their engagement in the implementation of the CMTS prototype, and to Eckhardt Holz for his detailed simulation study to analyse the behaviour of the CMTS service under various boundary conditions.

A large number of in-depth discussions with a lot of resulting stimuli have taken place during the CMTS design within Tenet research team at International Computer Science Institute and University of California at Berkeley. In particular, Prof. Domenico Ferrari as head of Tenet team and the group members Riccardo Gusella, Bruce Mah, Dinesh Verma, and Hui Zhang have provided very valuable suggestions during the preparation of this paper. This support is sincerely acknowledged by the authors.

Special thanks also go to Prof. David Anderson and Ramesh Govindan for their comments which helped to improve an earlier version of this paper.

9. References

[AHS90] D. Anderson, R. Herrtwich, C. Schaefer, "SRP: A Resource Reservation Protocol for Guaranteed-Performance Communication in the Internet", Int. Comp. Sci. Inst., Technical Report No. ICSI TR-90-006 (1990).

[BaM91] A. Banerjea, 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).

[Che88] G. Chesson, "XTP/PE Overview", 13th Conf. on Local Computer Networks, IEEE Comp. Soc. (October, 1988), 292-296.

[ChW89] D. R. Cheriton, C. L. Williamson, "VMTP as the Transport Layer for High-Performance Distributed Systems", IEEE Commun. Magazine, Vol. 27, No. 6 (1989), 37-44.

[CLZ87] D. D. Clark, M. L. Lambert, L. Zhang, "NETBLT: A High-Throughput Transport Protocol", ACM SIGCOMM Workshop on Frontiers in Comp. Netw.(1987).

[DDK90] W.A.Doeringer, D. Dykeman, M. Kaiserswerth, B.W. Meister, H. Rudin, R. Williamson, "A Survey of Light-Weight Transport Protocols for High-Speed Networks", IEEE Trans. on Commun., Vol. 38, No. 11 (1990), 2025-2039.

[FeV90] D. Ferrari and D. Verma, "A Scheme for Real-Time Channel Establishment in Wide-Area Networks", IEEE J. Sel. Areas in Comm. SAC-8 (April, 1990).

[GiG91] M. Gilge and R. Gusella, "Motion Video Coding for Packet Switching Networks: An Integrated Approach," SPIE Conf. on Visual Commun. and Image Processing, Boston (November, 1991).

P. Haskell, K. H. Tzou and T. R. Hsing, "A Lapped-Orthogonal-Transform Based Variable Bit-Rate Video Coder for Packet Networks," Int. Conf. on Acoustics, Speech and Signal Proc., Glasgow, Scotland, May 23-26, 1989.

[HSS90] D. Hehmann, M. Salmony, H.J. Stuettgen, "Transport Services for Multi-Media Applications on Broadband Networks", Computer Commun., Vol. 13, No. 4 (1990), 197-203.

[ITC91] Proc. Workshop on "Continuous Time Media", Information Technology Center, Carnegie Mellon University, Pittsburgh (June, 1991).

[Leg91] D. Le Gall, "MPEG: A Video Compression Standard for Multimedia Applications," Commun. of the ACM, Vol. 34, No. 4, (1991).

[LiH91] M. Liebhold, E. M. Hoffert, "Toward an Open Environment for Digital Video", Commun. ACM, Vol. 34, No. 4 (1991), 104-112.

[LPS91] T. F. La Porta, M. Schwartz, "Architectures, Features, and Implementation of High-Speed Transport Protocols", IEEE Network Magazine, Vol. 5, No. 3 (1991), 14-22.

[MoW91] M. Moran, B. E. Wolfinger, "Design of a Continuous Media Data Transport Service and Protocol", unpublished (1991).

[NRS90] A.N. Netravali, W.D. Roome, K. Sabnani, "Design and Implementation of a High Speed Transport Protocol", IEEE Trans. on Commun., Vol. 38, No.11 (1990), 2010-2024.

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

[Wat89] R. W. Watson, "The Delta-t Transport Protocol: Features and Experience", Proc. IFIP Workshop on Protocols for High-Speed Networks, North-Holland (1989), 3-18.

[WrT90] D. J. Wright, M. To, "Telecommunication Applications of the 1990s and their Transport Requirements", IEEE Network Magazine, Vol. 4, No. 2 (1990), 34-40.

[Zit91] M. Zitterbart, "High-Speed Transport Components", IEEE Network Magazine, Vol. 5, No. 1 (1991), 54-63.

[Zha91] L. Zhang, "Virtual Clock: A New Traffic Control Algorithm for Packet-Switched Networks," ACM Trans. on Computer Systems, Vol. 9, No. 2 (1991), 101-124.

10. Figures

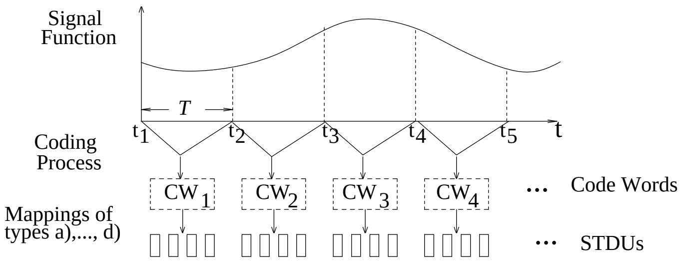

Figure 1: Coding of a stream and its mapping onto a sequence of STDUs

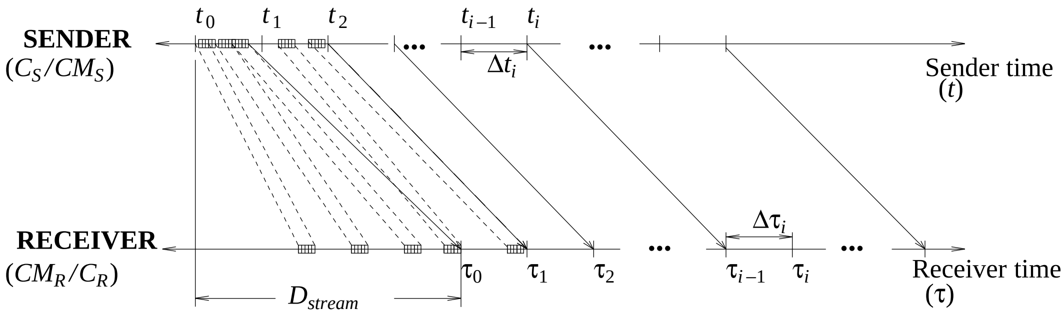

Figure 2: Basic timing during the transfer of the data corresponding to a stream

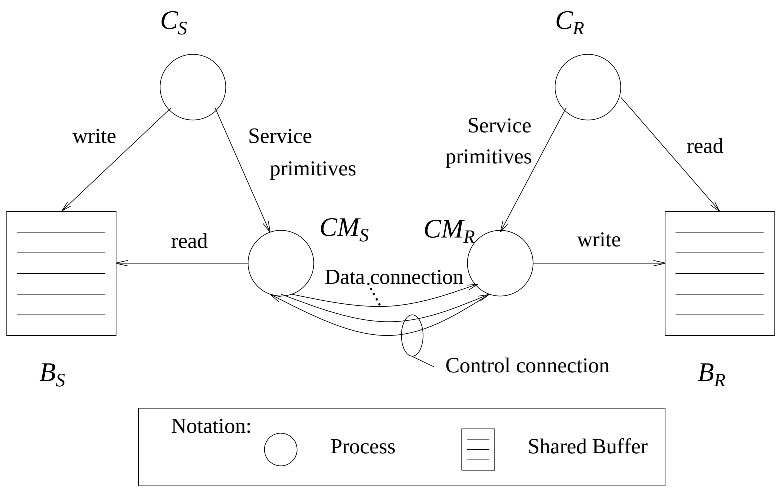

Figure 3: Basic components of CMTS implementation and illustration of their interactions

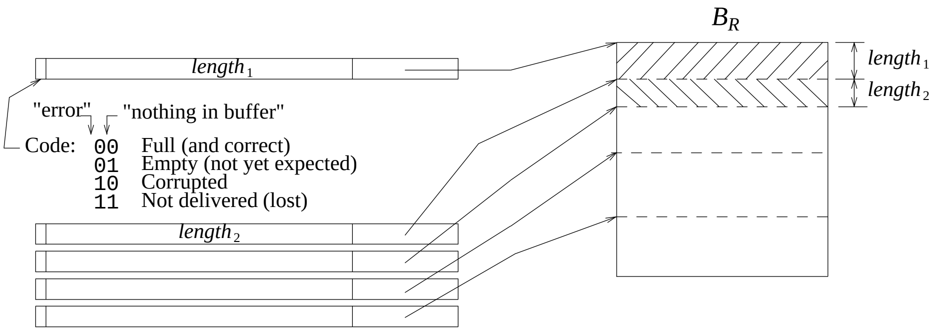

Figure 4: Shared buffer at the CM_R/C_R-interface and associated descriptors