Tenet Suite 1 and the Continuous Media Toolkit

Peter Staunton
staunton@tenet.CS.Berkeley.EDU

Tenet Group
University of California at Berkeley &
International Computer Science Institute

TR-95-028
June 1995

Abstract

The Continuous Media Toolkit (CMT) [1][2] is a flexible toolkit which facilitates development of local and distributed continuous media applications. Data transfer across a computer network is provided on a connectionless, best-effort basis using a network protocol called “Cyclic-UDP” [3]. A second set of network protocols, called “Tenet Suite 1” [4][5], has been designed to provide a simplex, unicast, connection-oriented service to realtime traffic in a packet-switched internetwork, with guaranteed performance in terms of data throughput, end-to-end delay, delay jitter, and loss rate. This report describes an extension to CMT which allows an application developer to employ the guaranteed network services of Tenet Suite 1.

Keywords: multimedia, continuous media toolkit, Tenet protocols, real-time communication, continuous media player.

1. Introduction

This report is organized as follows. In order to appreciate the issues involved in using Tenet Suite 1 and in extending CMT, it is important that the reader be familiar with both. To set the scene for the remainder of the report, section 2 presents both systems. The addition to the toolkit of support for JPEG decompression using the DEC J-Video hardware is described in Section 3. Motivation for this work is also given in that section. Section 4 describes the development of new toolkit objects which give the application developer access to the guaranteed network services of Tenet Suite 1. Section 5 describes the extension of a distributed network playback application, the Continuous Media Player, to make use of these new network toolkit objects. Section 6 offers suggestions for future work, and finally, Section 7 concludes the report.

2. Background

This section presents a brief description of CMT and of the Tenet Suite 1 network protocols. Familiarity with both will help greatly in understanding the work described in the later sections.

2.1 Overview of the Continuous Media Toolkit

CMT was developed by the Berkeley Plateau Multimedia Research Group at the University of California at Berkeley, led by Professor Lawrence A. Rowe. This is a freely distributed software package which facilitates the development of multimedia applications. It is a collection of object types which can serve as audio or video sources, audio or video devices, or intermediate objects which transfer or filter continuous media data. Support is provided for a wide variety of media formats, hardware platforms, and operating systems. The toolkit is based on Tcl/Tk [6]. Applications can be easily developed by writing short, simple scripts without any familiarity with compiled programming languages such as C or C++. This API empowers the developer to create, connect, control, and destroy objects with relative ease. This project used the first alpha release of version 3.0 of CMT (CMT3.0a1 released March 1995).

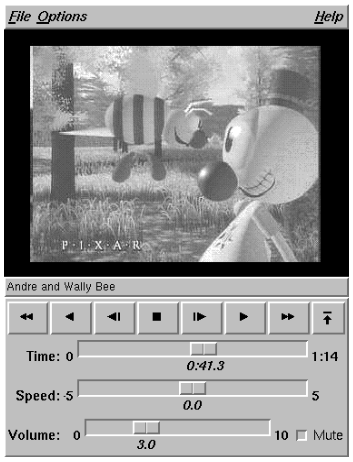

Figure 1: CM Player user interface.

Information is transferred from source object to destination object by means of a continuous flow of data called a stream. An application that plays movies, for example, may use parallel audio and video streams. In this case, each stream consists of a sequence of clips where each clip is a segment of data from an audio or video file. Clearly, objects need to be linked in some way to ensure that they operate in a synchronized fashion. This synchronization is achieved by means of clock objects (called logical time system "LTS" objects) to which all time-critical objects refer. The logical time of these clocks can progress at a rate faster or slower than real time to allow the user to "fast-forward" or "slow-forward" through a movie. Logical time can also advance at a negative rate to allow movie playback in reverse.

A number of sample applications accompany the toolkit. The most extensive of these is the Continuous Media Player (CM Player) [7]. This allows a user to select and view a movie stored on either a local or remote file server, or a combination of both. The graphical user interface is shown in Figure 1.

The interface controls are similar to those provided by a conventional video player, with support for playback either in forward or reverse at a speed of the user's choice. The user chooses a movie script by clicking on the "File" button and selecting from a file display box. A movie script comprises a list of audio or video clips. Each clip entry specifies media format and data location information. The "Options" button provides access to a set of menus which allows the user to select among alternative audio and video devices (e.g. hardware Motion JPEG, software-only Motion JPEG etc.).

2.2 Overview of Tenet Suite 1

Tenet Suite 1 is a set of network protocols developed by the Tenet Group at the University of California at Berkeley and at the International Computer Science Institute, led by Professor Domenico Ferrari. This protocol suite guarantees end-to-end network performance over a shared, packet-switched, data network to an application which has realtime requirements. This service is achieved through resource management, connection admission control, and appropriate packet service disciplines within the network routers. During connection establishment, the application describes its *traffic* in terms of parameters such as minimum inter-message time and maximum message size. It also specifies *performance* requirements in terms of end-to-end delay, delay-jitter, and loss rate. If the connection request is accepted, the protocol guarantees service and data transfer can take place. If, on the other hand, the requested performance cannot be guaranteed, the connection request is rejected and no data transfer can take place. This action is analogous to receiving a busy signal during an attempt to make a telephone call.

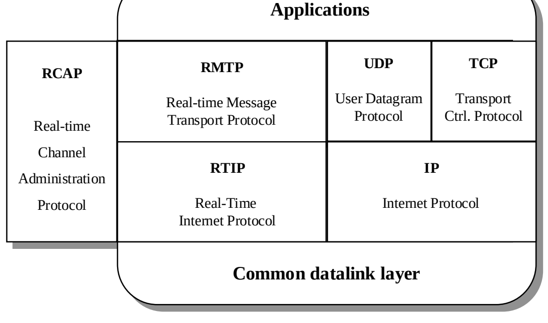

Figure 2: Network architecture of Tenet Suite 1.

The architecture of Tenet Suite 1, shown in Figure 2, consists of the network protocols RTIP, RMTP, and RCAP. Real-Time Internet Protocol (RTIP) [8] exists at the network layer and guarantees end-to-end packet delivery. Real-time Message Transport Protocol (RMTP) is a transport layer protocol that sits above RTIP, fragmenting and reassembling messages as required. Real-time Channel Administration Protocol (RCAP) [9][10] is responsible for channel establishment, status reporting, and channel teardown. As the figure shows, this protocol suite coexists with the Internet protocols TCP/IP and UDP/IP. Applications have simultaneous access to services from both.

3. CMT support for J-Video hardware

This section describes work done to allow CMT to exploit the image decompression services of the DEC J-Video hardware.

3.1 Motivation

Two types of compressed video are currently supported within CMT, namely Motion JPEG and MPEG.

Motion JPEG is simply a sequence of still images, each image having been compressed using the JPEG (Joint Photographic Experts Group) [11] standard for still image compression. Each frame can be viewed as being independent of all others since only intra-frame compression is performed.

MPEG (Motion Picture Experts Group), on the other hand, is a standard designed specifically for compression of video. Both intra-frame and inter-frame compression are performed, leading to a dependency between frames not found in Motion JPEG.

Compressed images are stored in files, and transferred, frame by frame, to the user's location where they are decompressed and displayed using a local X server.

CMT provides image decompression in both hardware and software [12]. On a DEC 5000 workstation running ULTRIX 4.2A, typical display rates of 1 frame every second and 1 frame every 2-3 seconds, were observed for software decoding of S/F sized (320x240) video using MPEG and Motion JPEG respectively. Image decompression requires intensive processing which, when implemented in software, introduces a large latency. This gives rise to a bottleneck in the decompression process, with frames being displayed at a rate bounded by the rate of image decompression. Hence, the low frame display rate observed above. All frames which cannot be decompressed in time are dropped. This effect is disturbing to the user, since jerky movement is displayed, rather than smooth video. Real-time software decompression can generate smooth motion if smaller images (e.g. 160x120) or faster processors (e.g. current generation RISC processors that operate at 100 SpecInt92s) are used.

In order to construct an impressive demonstration of the use of Tenet Suite 1 with CMT, it is necessary to provide the user with smooth video in whatever environment this suite of network protocols is supported. For the purpose of this project, this environment is a cluster of DEC 5000 workstations. The available video hardware is the DEC J-Video board, which will facilitate a higher frame display rate since decompression is done in hardware as opposed to software.

The J-Video board was an engineering prototype for the DEC J300 product. It has a C-Cubed processor chip which can decode 60 fields per second. The task at hand here, therefore, is to provide support within CMT for JPEG decompression performed by the J-Video board.

3.2 Solution

Figure 3 shows the architecture of the J-Video CMT object. “jvdriver” is the driver for the J-Video board. It serves as the interface to the underlying hardware which reads a compressed image from memory and writes the decompressed image back into memory. This driver must run as a separate process. Commands are communicated to the driver via a library of function calls called “JvDriverInt”. In an effort to prevent future confusion, it should be noted that there are two drivers with the same name (jvdriver) available, one of which is compatible with this library, a second which is not. The program “jvdriver” must be executed before one starts an application that relies on its services.

“MjpegPlay” is the CMT object to which compressed JPEG images are passed. This object is responsible for decompression and subsequent display of the image. One or more devices would usually be available to provide JPEG decompression services to this object. For example, such a device implemented in software is provided with the toolkit. For our purposes, we require that the J-Video hardware provide these services. We shall refer to the corresponding device as “MjpegJvideoDevice” following the naming convention of the toolkit.

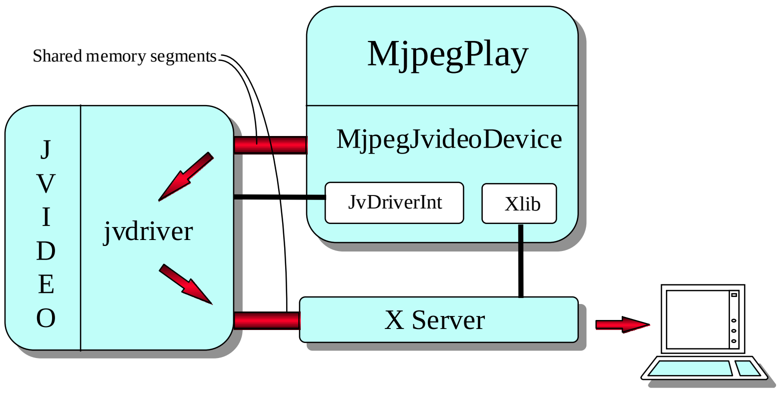

Figure 3: JPEG Decompression with J-Video board.

A pair of shared memory segments is indicated in Figure 3. A shared memory segment is an area of memory which is shared by two or more processes. This shared property obviates the need to involve the CPU in copying data from the address space of one process to that of another since they both have direct access to this memory, leading to reduced latency. For a shared memory segment to be created, one process must first reserve this area of memory. Other processes that require access must perform an “attach” operation to map the memory into their logical address spaces.

The X server shown is a standard server with the addition of the MIT shared memory extension. This extension allows the X server to display images stored in shared memory segments. Communication with the X server is by function calls from the library “Xlib”.

Before image decompression can occur, “jvdriver” must reserve these two shared memory segments, one from which it can read compressed JPEG data, the other to which it can write decompressed image data. The former is shared with the object “MjpegPlay”; the latter is shared with the X server. Unfortunately, “jvdriver” insists on reserving these segments itself rather than being able to attach to previously reserved segments. This inflexibility proves to be inconvenient since it requires that the compressed image data be copied into the shared memory segment each time a frame needs to be decompressed. Previous work has shown, however, that the time taken for this copy is small relative to the time taken by the J-Video hardware to decompress an image. The impact on the image display rate is therefore not significant.

“jvdriver” also requires that parameters such as image dimensions and JPEG quantization factor be defined as part of an initialization process. Subsequent change of these parameters requires the device to be re-initialized.

The course of events is as follows for each individual frame. Object “MjpegPlay” receives a pointer to the location of the compressed image data. This data may come from a file object or from a network destination object. “MjpegPlay” copies the data from this location to the memory segment shared with “jvdriver”. Using a call from the “JvDriverInt” library, a command is passed to “jvdriver” to commence JPEG decompression. “jvdriver” reads the compressed data from this segment, decompresses it, and writes the resultant image data to the memory segment it shares with the X server. Meanwhile, “MjpegJvideoDevice” blocks awaiting completion of the decompression operation. When the driver signals completion of decompression, “MjpegJvideoDevice” commands the X server to update the display with the image from the shared memory segment.

Finally, when the “MjpegPlay” object is destroyed, commands are passed to “jvdriver” to relinquish the shared memory segments and to the other processes to detach from it.

4. New CMT objects for Tenet Suite 1

This section takes a closer look at “Cyclic-UDP” [3], the set of network protocols already supported by CMT. The functional requirements of the toolkit’s network objects are discussed, and consideration given to those of new objects that use the protocols of Tenet Suite 1. The implementation of these new objects is described. Details are given on how these can be used by the application developer.

4.1 Cyclic-UDP

CMT is based on Tcl/Tk [6], a scripting language and interface toolkit developed by Professor John Ousterhout at the University of California at Berkeley. A number of commands have been added to the basic Tcl/Tk interpreter which allow creation and manipulation of media objects, and facilitate distributed programming. When we execute this extended interpreter, we create what is called a “CM process”.

CMT is a collection of objects or building blocks which can be used to construct an application. If the source of the data (e.g. a file object) is local to the data destination (e.g. a play object), a single CM process is sufficient. If, on the other hand, the source is not on the same host as the destination, then a CM process is required on both hosts. An example of this is shown in Figure 4.

Here, we have a user at host B playing a Motion JPEG movie located at host A. Image data is passed from source (Motion JPEG file object) to destination (Motion JPEG play object) via the intermediate network objects “Packet Source” and “Packet Dest”. We omit to show LTS clock objects on hosts A and B that synchronize the file and play objects. A control object on host B allows the CM process on B to pass Tcl commands to a control object on A over a TCP/IP connection. These commands would typically be of the type “object create”, “object configure”, “start play”, “stop play”, “object destroy”, and so forth.

Let us now focus on the network objects “Packet Source” and “Packet Dest”. These communicate using a protocol called “Cyclic-UDP”. This is a best-effort, no-guarantee protocol based on the Internet transport layer and network layer protocols UDP/IP.

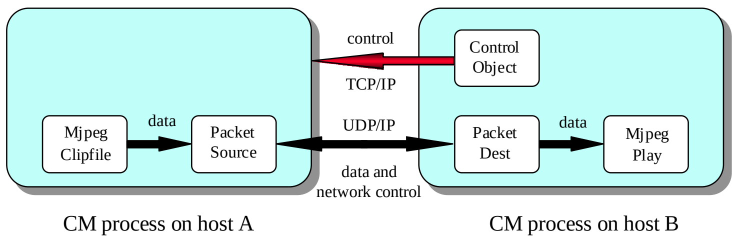

Figure 4: Distributed CM application with Cyclic-UDP.

Cyclic-UDP performs the following functions:

- **Data transfer**: payloads of continuous media data with prepended packet headers are sent in packets from “Packet Source” to “Packet Dest”.
- **Fragmentation**: frames of continuous media data are fragmented into packets of a fixed maximum size, typically 8000 bytes. “Packet Source” fragments each frame into a number of packets and “Packet Dest” performs reassembly to the original frames.
- **Detection of packet loss**: “Packet Dest” monitors which packets have and have not been received and detects if a packet has been lost. It may then request the “Packet Source” to resend the packet. This request is communicated by means of a UDP datagram sent in the direction opposite to the flow of data which explains the two-ended arrow between the two network objects in the diagram of Figure 4 above.
- **Flow control**: measurements and calculations are made which estimate available bandwidth, end-to-end delay, delay jitter, and packet loss rate between the network objects. These estimates are communicated by “Packet Dest” to “Packet Source”, again by UDP datagrams, so that it can adaptively regulate packet transmission.
- **Prioritization**: it is possible to place greater importance in a media-specific way on some frames relative to others, so that any effect of increased network load will have the least impact on the quality of the user’s display. An example is to place I frames at a higher priority level than P and B frames for transmission of an MPEG data stream.

4.2 Design of the Tenet objects

In order to add the capability of transferring data using Tenet Suite 1 to CMT, we must design new network objects which can substitute for “Packet Source” and “Packet Dest” above. We call these new objects “Tenet Source” and “Tenet Dest”. The application of Figure 4 now appears as shown in Figure 5.

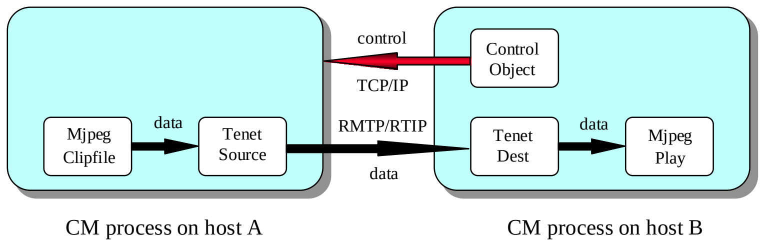

Figure 5: Distributed CM application with Tenet Suite 1.

Let us examine each of the functions provided by Cyclic-UDP, and consider whether these are required in our new Tenet objects.

- **Data transfer**: obviously, we still need to transfer continuous media data from “Tenet Source” to “Tenet Dest”.
- **Fragmentation**: RMTP, the transport layer of Tenet Suite 1, performs any necessary fragmentation and reassembly of messages into packets and of packets into messages respectively. Prior fragmentation at the user level of the application is unnecessary, so this function is not required when using the Tenet objects.
- **Detection of packet loss**: during connection establishment, a statistical or deterministic guarantee of message loss rate is made by RCAP. If we request a loss rate of zero with probability of one, detection of packet loss is not an issue since none should occur. This is, therefore, another function which will not be needed in the Tenet objects.
- **Flow control**: during connection establishment, we specify to RCAP a number of parameters that describe the traffic to be carried. `Xave` is the minimum average inter-message time. It is the inverse of the maximum message rate. `I` is the interval over which this average is to be calculated. `Xmin` is the minimum inter-message time. `Smax` is the maximum message size. If RCAP accepts the channel request, we are guaranteed sufficient throughput provided our traffic does not step outside the bounds of this traffic description. Flow control, then, is no longer an issue since we do not need to regulate the transmission rate at the “Tenet Source”. Estimates of bandwidth, end-to-end delay, and delay jitter are no longer required. In fact, we need not feed back any network control information from “Tenet Dest” to “Tenet Source”. We can represent the unidirectional information flow between these by a single-ended arrow as in Figure 5.
- **Prioritization**: since RCAP guarantees us sufficient bandwidth for our continuous media traffic, prioritization of frames is of no benefit since we expect that all frames will be delivered and be delivered on time.

4.3 Use of the Tenet objects

The Tenet objects have been implemented using the C programming language [13][14]. New Tcl commands have been added to the toolkit to allow the application developer to create, configure, control, and destroy these objects. The developer should require no knowledge or familiarity with the underlying C code.

4.3.1 Object creation

Tenet objects are created using the following lines of Tcl in a script:

```
1.    set td [tenetDest “”];
2.    set ts [$cm create tenetSrc];
```

The first line creates a “Tenet Dest” object on the local host and stores the name in variable *$td*. A port is registered with RCAP using the function `RcapRegister` and the port number is stored as an instance variable of the object. A child process is spawned which blocks on the function call `RcapReceiveRequest` while it waits for an establish channel request on that port. It is because this call blocks that we require an extra process. A pipe is also created so that information can be passed back to the parent from the child when this request arrives. The parent process is free to return to the Tcl script where it should later send a command to the “Tenet Source” to attempt channel establishment to this host and port number.

Line two creates a “Tenet Source” object on the remote host and stores the name in variable *$ts*. *$cm* refers to a control object which passes commands to the remote CM process as shown in Figure 5.

4.3.2 Channel establishment

Since Tenet Suite 1 provides connection-oriented services and guarantees on a per-connection basis, a channel must be established before data transfer can take place. The following lines of Tcl code will establish such a realtime channel. Lines 3-9 can be implemented using a single configuration command. They have been shown separately here for ease of reference.

```
3.    $ts config -xave 40;
4.    $ts config -xmin 20;
5.    $ts config -I 1000;
6.    $ts config -smax 10000;
7.    $ts config -delay 50;
8.    $ts config -jitter 5;
9.    $ts config -dest [$td address];
10.   $td connect;
```

The second half of line nine returns the address at which the “Tenet Dest” expects to receive an establish channel request. This address consists of an IP address, followed by a RCAP port number. The rest of this line commands the “Tenet Source” to make a connection to this destination. It does this by calling the function `RcapEstablishRequest` which, if successful, returns the logical channel number. It then associates a socket with this logical channel by means of the `setsockopt` function call. If unsuccessful, an error is returned to the application.

The traffic description parameters are specified in lines three through six. The values of `Xave`, `Xmin` and interval `I` should be specified in milliseconds. RCAP accepts these parameters in a different format where the integer 65536 signifies 1 second but the C code for the “Tenet Source” object will do the necessary conversion. `Smax` is the maximum message size in bytes. If values are not specified in advance of the channel request, default values of 50, 25, 1000, and 10000 are assumed for `Xave`, `Xmin`, `I`, and `Smax` respectively.

Channel performance parameters are set in lines seven and eight. These are the end-to-end delay bound and the delay jitter bound, both in milliseconds. If not specified, these default to 100 and 5 respectively. Other performance parameters passed to RCAP are the end-to-end statistical delay probability, end-to-end statistical "no-drop" probability, and end-to-end statistical jitter probability. In this implementation, values of one have been used for each of these to give a fully deterministic channel. These values are unalterable by, and invisible to, the application developer. It would be straightforward, however, to add C code to the "Tenet Source" object to provide the developer with access to these.

If the `RcapReceiveRequest` call above is successful, the child process approves the request by calling the function `RcapEstablishReturn`. The success status of the `RcapReceiveRequest` call and the logical channel number of the established channel are written to the pipe before the child process dies.

Finally, line ten tidies up at the destination in the following way after the attempt to establish a channel. Information from the child process is retrieved from the pipe. If a channel has been established, we associate a socket with this logical channel using a `setsockopt` call as we did at the source. A Tcl/Tk file handler is created to provide a callback when data arrives at this socket. If a channel has not been established, an error message is returned to the application.

4.3.3 Transfer of continuous media data

An object in CMT sends data to a local object by passing it a scatter buffer list which is a list of pointers to buffers where the actual data is stored. There exists a scatter buffer list for each frame of data.

When a "Tenet Source" object has frames to send, it prepends a small header to each frame and sends them, individually, to the socket which is associated with the realtime channel. This header contains a frame number, the frame size, and a timestamp of the time of transmission. The size of this header, 16 bytes, is much smaller than the 68-byte header prepended to each packet in Cyclic-UDP. This size reduction is an indication of the greater functionality of the network objects in Cyclic-UDP.

At the destination, a "Tenet Dest" object will receive a callback from a Tcl/Tk file handler whenever there is incoming data to be read from the logical channel. The header is removed and the remaining data is written to a buffer. A scatter buffer list is constructed and passed to the next object which is then free to manipulate the data however it wishes.

Once we have specified traffic characteristics to RCAP during channel establishment, it is important that we do not violate these parameters so that the guarantees can be fulfilled. Recall that `Xmin` is the minimum inter-message arrival time. We must ensure that the application waits for at least this length of time before sending another frame. This constraint is achieved using a Tcl/Tk timer handler as follows. If the "Tenet Source" object sends a frame, it checks to see if any further frames are queued up to be sent. If there are, a Tcl/Tk timer handler is created. This handler will wait the period of time given by `Xmin` (during which time other operations can be performed), after which a callback will be made to the function to send another frame. This function will check for other waiting frames as before, schedule a callback if necessary, and so on until all the frames have been sent or the queue has been cleared for some other reason.

Once the objects have been created and linked both to themselves and to clock objects, the application developer does not need to be aware of the intricacies of how frames are stored in buffers, sent to sockets, passed to objects in scatter buffer lists and so forth. This is all implemented in C code, to which the developer should be oblivious.

4.3.4 Channel teardown

Tenet objects are destroyed by the following lines of Tcl code:

```
11.   $td destroy;        ## $td is a Tenet Dest object.
12.   $ts destroy;        ## $ts is a Tenet Source object.
```

When the “Tenet Dest” object is destroyed, the channel is closed using the `RcapCloseRequest` function call. The RCAP port is also released using the `RcapUnregister` call. Data sockets used in “Tenet Dest” and “Tenet Source” objects are closed when these objects are destroyed.

4.3.5 Interaction of Tenet objects with other objects

It has already been described how one Tenet object transfers data to the other but not how these interact with other objects in the toolkit. This is achieved in the standard CMT fashion. Take for example the application of Figure 5. To arrange that a Motion JPEG file object *$file* sends data to a “Tenet Source”, and that a Motion JPEG play object *$play* receives data from the corresponding “Tenet Dest”, the following lines of Tcl code can be used:

```
13.   $file config -outCmd “$ts accept”;   ## $ts is a Tenet Source object
14.   $td config -outCmd “$play accept”;   ## $td is a Tenet Dest object
```

These commands define the output command of objects. When an object has data to pass on to another object, it calls this command.

Please note that the lines of code above have been numbered for reference purposes only and do not together form a complete application in any shape or form.

4.4 Simpler implementation

The network objects of Cyclic-UDP have much greater functional responsibility than the Tenet objects. This is reflected in the number of lines of code used to implement them, 3716 for those of Cyclic-UDP, as opposed to 2271 for the Tenet objects. One could claim, therefore, that an advantage of the Tenet approach is that the distributed application need not be as complex since there is greater knowledge of, and certainty in, the network services available.

5. Tenet objects in the Continuous Media Player

The previous section described the addition of Tenet objects to CMT which allows the developer to build an application that uses the Tenet Suite 1 of network protocols for data transfer. To test and demonstrate the use of these objects, the application CM Player has been extended as described in this section.

5.1 Configuration of network

Before applications which use the Tenet Suite 1 can be executed, the network must first be prepared. The transport layer and network layer protocols, RMTP and RTIP, must be installed in the kernel at each node of the proposed route. An RCAP daemon must be running on each node, using a configuration file that is appropriate to the architecture of the node and the intended route.

5.2 Traffic and performance parameters

Section 4.3 described the RCAP traffic and performance parameters of the “Tenet Source” object which the developer can set using Tcl commands. The question remains, what values should the developer use for these parameters?

5.2.1 RCAP traffic parameters

Recall that `Xave` is the minimum average inter-message time. Since each message passed to RMTP holds a single frame of data, `Xave` should be the minimum average inter-frame time. This value is the inverse of the maximum frame rate.

In CM Player, `Xave` is set using the following lines of Tcl code in the case of a MPEG or Motion JPEG video stream:

```
15.    $file config -maxFrameRate 30;
16.    $ts config -xave [expr 1000/30];
```

The variable *$file* refers to a file object whose output frame rate is set in line 15. `Xave` is then configured to be the inverse of this frame rate. The factor of 1000 is used to convert `Xave` to milliseconds. The variable *$ts* refers to a “Tenet Source” object as before.

With MPEG and Motion JPEG, each frame holds a single image or video sample. The situation is different for an audio stream where a sample is only one or two bytes of data. In this case, many samples of data are collected together to form a frame. Frames are passed to the next object with a frequency given by the “cycle time”. `Xave` can, therefore, be set equal to this cycle time. In CM Player, `Xave` for an audio stream can be set to one second using the following lines of Tcl code:

```
17.    $afile config -cycleTime 1.0;
18.    $ts config -xave 1000;
```

The variable *$afile* of line 17 refers to an audio file object whose cycle time is set to one second. Line 18 sets `Xave` to the same value. Again a factor of 1000 is used to express `Xave` in milliseconds.

Recall that traffic parameter `I` is the interval over which `Xave` is calculated. In CM Player, we do not set this value, so it assumes its default value of 1000 milliseconds.

Recall also that the parameter `Xmin` is the minimum inter-message time. When the “Tenet Source” object sends a frame, if it has further frames waiting, it creates a Tcl/Tk timer handler to call itself back after `Xmin` milliseconds. Since the application may be performing an atomic task at the scheduled time, the callback may not always be made exactly on time. It is therefore not recommended that `Xmin` be set equal to `Xave` since these slightly late callbacks could give rise to frame loss in the “Tenet Source” object. Measurements for a message rate of 30 frames per second have shown that if `Xmin` is set to 80% of `Xave` or lower, all frames are transmitted. In the script for CM Player, `Xmin` is set equal to 80% of `Xave` for this reason. An alternative solution would be to allow the developer to specify any `Xmin` value up to `Xave.` The “Tenet Source” object would then reduce this value before giving it to RCAP.

Finally, Smax is the maximum message size. Since each message is one frame of data, this value is the maximum frame size. Before passing this value to RCAP during connection establishment, the “Tenet Source” object will add the size of the frame header which was discussed in section 4.3.3.

In the case of Motion JPEG and MPEG video streams, the following lines of Tcl code show how we set the value of Smax:

```
19.    $file addSegment 2 3 -start 5 -end 6;
20.    $ts config -smax [$file maxSize];
```

The maximum frame size is stored as part of the Motion JPEG or MPEG file object for each segment which makes up the movie. These objects have been extended so that, by passing it the single parameter “maxSize” as in the second part of line 20, the maximum frame size over all segments of the movie is determined and this value is returned. The remainder of this line sets the Smax parameter of the “Tenet Source” object to be equal to this value. Line 19 has been included to point out that data segments which make up the movie must be added to the file object before it can return an accurate value for the maximum frame size.

In the case of an audio stream, the data rate and cycle time are stored for each audio file object. The data rate is a function of the audio sampling rate of the encoded audio, and the number of bytes per sample. The frame size can be calculated by taking the product of the data rate and the cycle time. For example, if the sampling rate were 8000 samples per second, each sample being a single byte, the data rate would be 8000 bytes per second when audio is played back at normal speed. Furthermore, if the cycle time is 0.5 seconds, then the maximum frame size should be 4000 bytes.

```
21.    $ts config -smax [$afile maxSize];
```

This line of Tcl code shows how the Smax parameter is set for an audio stream. The audio file object has been extended to return this maximum frame size when passed the single parameter “maxSize”. The Smax parameter of the “Tenet Source” is then set equal to this value.

5.2.2 RCAP performance parameters

As described in section 4.3.2, the developer may configure the “Tenet Source” object with desired values for end-to-end delay and delay jitter bounds. In CM Player, these values have not been specified and will, therefore, assume their default values of 100 and 5 milliseconds respectively.

5.3 Environment variable CM_HOSTNAME

The situation can arise where the user’s host has more than one network interface and it is preferred that RCAP channels use one of these instead of the others.

For example, we have used `faith.CS.Berkeley.EDU` and `truth.CS.Berkeley.EDU` during development. These two hosts are connected to both an FDDI ring and an Ethernet. The user’s host can be called `faith` or `faith-fddi` by `truth` to indicate which route should be used. These two host names (`faith` and `faith-fddi`) also have different IP addresses to distinguish the two network interfaces on `faith`.

By default, a source network object will send data to the IP address which corresponds to the user's host name as returned by the function `gethostname`. In our example, this address is the IP address of `faith.CS.Berkeley.EDU` which is 128.32.33.105. We prefer, however, that RCAP channels be established over the FDDI route instead. So that this can be achieved, an environment variable *CM_HOSTNAME* has been introduced. This variable is set to `faith-fddi.CS.Berkeley.EDU` on the user's host before CM Player is executed. The "Tenet Dest" object will use the name given by this environment variable if it exists and return the corresponding IP address which in this case is 192.107.102.70. If the environment variable does not exist, the host name returned by `gethostname` is used instead.

5.4 Choice of network protocols

As previously discussed, Tenet Suite 1 provides a number of guarantees to realtime traffic on a per-connection basis. Up until now, however, we have assumed that the available network resources have always been ample to provide us with whatever level of service is requested. If such resources are not available, no guarantees are made and the request for channel establishment is denied.

The question then arises, what is the application to do if the request is rejected? A few different options are possible. One such option is to reduce the requested quality of service. For example, one could specify a lower frame rate by increasing the `Xave` and `Xmin` traffic parameters. The source file object would be configured to output a lower frame rate to reflect this change. Also, one could request less stringent end-to-end delay and delay jitter bounds. In CM Player, delay jitter is not normally an important factor since frames, when received by a play object, are usually not played immediately. They are typically received in advance of their correct play time and stored until that time arrives.

In fact, here lies another advantage of the Tenet protocols. Memory is normally used within the application to smooth out delay jitter. One could reduce the amount of memory used by the application for this purpose by relying on the delay and delay-jitter bounds guaranteed by the network. Knowledge of the end-to-end delay and the corresponding jitter allows a reduction in the "send-ahead time" i.e. the amount of time in advance of the play time that the source sends a frame. In this way, a guaranteed network service can reduce the startup latency of an application compared with a best-effort network service. In this case, the guaranteed jitter bound becomes important. If one increases it to improve the chances of channel acceptance, one must also modify the send-ahead time appropriately.

A second option is to revert to an alternative set of network protocols. Since the network cannot provide us with the guarantees we wish for, why not switch to a set of network protocols which function on a best-effort, no-guarantee basis such as Cyclic-UDP? Without guarantees, Cyclic-UDP will estimate values for available bandwidth, end-to-end delay, and delay jitter. It will use this information to adaptively tune itself for optimal fidelity to the user, given the current condition of the network.

The power to change to a different network protocol suite could be given to a user in the form of a switch or a menu as part of the user interface. Applications of the future may incur different billing charges dependent on whether there is guaranteed or best-effort service. The user may therefore be required to make a conscious decision between them. In CM Player, we have added an extra option

"Network Protocols" to the "Options" menu which allows the user to select between the default selection "Tenet Suite 1" and an alternative "Cyclic UDP" as illustrated in Figure 6. By default, CM Player uses Tenet objects for data communication across a network. At the user's request, the application can shift to the packet objects of Cyclic-UDP. The addition of this option required only 8 extra lines of Tcl code and was therefore an inexpensive modification.

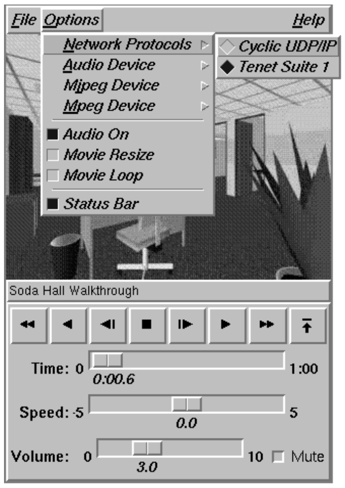

Figure 6: Choice of network protocols in CM Player.

Perhaps the protocol suite is of no concern to the users and should be invisible to them. One might argue that the user interface should be kept as simple as possible and that this option is one complication with which the user should not have to deal. If so, the decision is then the responsibility of the application. It should, by negotiation with the network, be able to decide if a guaranteed or a best-effort network paradigm is most suitable at any given time and adjust its data communication mechanisms appropriately.

One final possibility is that a combination of two sets of network protocols be used. As mentioned earlier, Cyclic-UDP is able to prioritize frames in a media-specific fashion so that more important frames experience a higher effective network throughput. A continuous media stream could be subdivided in this way into separate streams of different priority. Protocols that provide guarantees could be used for the higher priority sub-streams while sub-streams of a lower priority could be carried by a different protocol suite on a best-effort basis.

6. Future work

This section offers suggestions for future work related to this project.

6.1 Measurements

Measurements should be taken using CM Player to compare the performance of Tenet Suite 1 versus that of Cyclic-UDP under loaded and unloaded network conditions. The objective would be to validate further the guaranteed service that Tenet Suite 1 claims to provide. A suitable location for such tests would be the Tenet Group’s PC testbed which is a mesh of two-host Ethernet segments connecting hosts running BSDI BSD/OS 2.0 UNIX. The intention was to include details of these measurements in this project but at the time of project completion, the PC implementation of Tenet Suite 1 is not yet stable enough for this purpose.

6.2 Application to demonstrate the Tenet protocols

Now that Tenet network objects have been added to CMT, it would not be difficult to develop an application which would play two movies from the same remote source on the user’s workstation. Two streams of data would flow in parallel, one for each movie. One would use Tenet network objects, the other Cyclic-UDP packet objects. Cross-traffic would then be introduced to load the network, letting the user easily compare the fidelity of video and audio provided by the two sets of network protocols. Whereas the measurements of Section 6.1 above would be a scientific, quantitative validation of the Tenet protocols, the demonstration application described here would be best suited to public exhibitions of the Tenet Group’s work.

6.3 CMT objects for Tenet Suite 2

The suite of “realtime” network protocols we have used here, Tenet Suite 1, provides a unicast data service. Many continuous-media applications, however, will require performance guarantees in a multicast paradigm rather than a unicast one. Video-conferencing with many participants is one such example. The second suite of protocols from the Tenet Group, called “Tenet Suite 2” [15][16], will support multiparty communication with guarantees on throughput, end-to-end delay, and delay jitter bounds.

Network protocols that employ feedback of control information from the destination to the data source are said to operate in a *closed-loop* fashion. Transport Control Protocol (TCP) and X.25 are examples of this, as is Cyclic-UDP. Such protocols are difficult to extend for use in the multicast paradigm since the source can soon get overloaded filtering, and acting upon, feedback information as the number of destinations increases.

Protocols that operate in an *open-loop* fashion, conversely, do not require feedback of control information. In a network that does not offer guarantees but transfers data on a best-effort basis, reliability can be achieved by means of forward error correction mechanisms. On the other hand, network protocols which do offer guarantees, such as the Tenet suites, should not require feedback since the source has already received assurances regarding the level of service provided by the network and the ability of the receiver to absorb the traffic directed to it, even in the worst case.

In this project, network objects in CMT that operate in a closed-loop fashion have been replaced by “Tenet objects” that operate in an open-loop fashion. It has been shown that, if the network provides guarantees to the data source, continuous media applications such as CM Player can transfer data to a destination without the need for feedback control. Now that this work has been done, the task of developing toolkit objects for multicast communication should be much easier. Just as our Tenet objects have used Tenet Suite 1 protocols, the new objects could use the services offered by the next suite of protocols, Tenet Suite 2. In fact, most of the code could be replicated in the new objects, a notable exception being the replacement of the code which interfaces with RCAP with that which, instead, would interface with the new control protocol RCAP2. The extension of the application CM Player to send multicast data streams to many users, rather than a unicast stream to a single user, should be straightforward.

7. Conclusion

Experiences with the Tenet Suite 1 network protocols and the Continuous Media Toolkit have been described. New network objects were added to the toolkit which can substitute for those of Cyclic-UDP, allowing the application developer to easily employ the guaranteed network services of Tenet Suite 1. CM Player, a continuous-media application which allows a user to view a movie stored at a remote location, has been successfully extended to include these new network objects. The advantages of an application having a selection of network protocols at its disposal have been discussed, providing data transfer on either a best-effort or a guaranteed basis, the choice being influenced by the condition of the network at that time. Support for JPEG decompression using the J-Video board has also been added to the toolkit to allow smooth motion video in a development environment where the network protocols of Tenet Suite 1 are available. Finally, suggestions for further work which this project makes possible have been made, the most exciting of which is the addition of multicast stream support to CMT by using the services of the emerging set of realtime protocols in Tenet Suite 2.

Acknowledgments

I would like to thank Professor Domenico Ferrari for his friendly guidance during this work. I am grateful to Professor Lawrence A. Rowe who provided valuable feedback for this report. Thanks must also go to Tenet Group members Wendy Heffner, Bruce Mah and Ed Knightly who have been generous with their time when I experienced difficulty. Finally, I must acknowledge my parents who have always been a source of encouragement and moral support.

References

[1] K. Patel, *Introduction to CMT*, document to accompany the Continuous Media Toolkit, ftp://mm-ftp.cs.berkeley.edu/pub/multimedia/cmt/Hidden/cmt-3.0.a2.tar.Z (1995).

[2] L. A. Rowe, *Continuous Media Applications*, Presented at Multipoint Workshop held in conjunction with ACM Multimedia '94, San Francisco, CA, (Nov. 1994).

[3] B. C. Smith, *Implementation Techniques for Continuous Media Systems and Applications*, Ph.D. thesis in Computer Science, University of California at Berkeley, (September 1994).

[4] D. Ferrari, A. Banerjea, and H. Zhang, *Network Support for Multimedia - A Discussion of the Tenet Approach*, Computer Networks and ISDN Systems, vol. 26, pp. 1267-1280, (1994).

[5] A. Banerjea, D. Ferrari, B. Mah, M. Moran, D. Verma, and H. Zhang, *The Tenet Real-Time Protocol Suite: Design, Implementation, and Experiences*, Technical Report TR-94-059, International Computer Science Institute, Berkeley, CA, (November 1994).

[6] J. K. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley (1994).

[7] L. A. Rowe and B. C. Smith, *A Continuous Media Player*, Proc. 3rd. Int. Workshop on Network and OS Support for Digital Audio and Video, San Diego CA (November 1992).

[8] D. Verma and H. Zhang, *Design Documents for RTIP / RMTP*, unpublished, University of California at Berkeley and International Computer Science Institute, Berkeley, CA (May 1991).

[9] A. Banerjea and B. A. 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).

[10] A. Banerjea and B. A. Mah, *The Design of a Real-Time Channel Administration Protocol*, unpublished, University of California at Berkeley and International Computer Science Institute, Berkeley, CA (May 1991).

[11] W. B. Pennebaker and J. L. Mitchell, *JPEG Still Image Data Compression Standard*, Van Nostrand Reinhold (1992).

[12] K. Patel, L. A. Rowe and B. C. Smith, *Performance of a Software MPEG Video Decoder*, Proc. ACM Multimedia '93, Anaheim CA (August 1993).

[13] W. Kernighan and D. M. Ritchie, *The C Programming Language*, Prentice-Hall (1978).

[14] W. R. Stevens, *UNIX Network Programming*, Prentice-Hall (1990).

[15] A. Gupta, W. Heffner, M. Moran, C. Szyperski, *Network Support for Realtime Multi-Party Applications*, Fourth International Workshop on Network and Operating System Support for Digital Audio and Video, Lancaster, England, (November 1993).

[16] R. Bettati, D. Ferrari, A. Gupta, W. Heffner, W. Howe, M. Moran, Q. Nguyen, R. Yavatkar, *Connection Establishment for Multi-party Realtime Communication*, Proceedings of the Fifth International Workshop on Network and Operating System Support for Digital Audio and Video , Durham, NH, (April 1995).