Cyclic-UDP: A Priority-Driven Best-Effort Protocol

Brian C. Smith
Computer Science Department
Cornell University
bsmith@cs.cornell.edu

Abstract

This paper describes *cyclic-UDP*, a best-effort network protocol for audio and video file servers. Cyclic-UDP is divided into two layers, a *prioritization layer* and a *transmission layer.* The prioritization layer exploits properties of the media (e.g., audio or video) and its encoding (e.g., JPEG or MPEG) to prioritize packets. The transmission layer delivers the prioritized packets. The probability of successful packet delivery is proportional to the packet's priority. Combining these two layers produces a best-effort delivery mechanism that provides high fidelity reconstruction at the receiver despite network congestion and packet loss. Cyclic-UDP's prioritization algorithms, error correction scheme, flow control mechanism, and the results of experiments using cyclic-UDP are presented.

1. Introduction

The effective integration of audio and video (*continuous media,* or *CM*) data into the desktop computer environment will change the way we interact with computers as fundamentally as the shift from alphanumeric terminals to graphical user interfaces. To realize this vision, the problems of representing, storing, and transporting CM data must be solved. This paper describes a best-effort protocol for CM file servers called *cyclic-UDP (CUDP).*

To understand CUDP, one must understand how CM file servers are typically built. These servers typically serve a set of clients using *round-robin*, or *cyclic*, scheduling [11, 8]. That is, they periodically service each client by reading a sequence of video frames or audio samples (called *media units*) from a disk into a buffer. The buffer is then sent to the client, and transmission must be completed before the next service time. The important point is that the transmission protocol has the length of a cycle to transmit a group of media units to the client.

To send the buffer to a client, many CM file servers use a best-effort network transmission protocol such as UDP [6]. With such a protocol, some media units may be lost. Since buffer overflow is a common cause of packet loss, consecutive media units are often dropped, leading to an annoying stop-and-go effect in video and dropouts or "pops" in audio. Worse yet, depending on the encoding, the loss of some media units can have a precipitous effect. For example, in MPEG encoding, some video frames are encoded independent of other frames (I-frames), while others (P-frames and B-frames) are encoded as differences from these so-called reference frames [7]. If a reference frame is lost, the dependent frames cannot be decoded.

These problems can be overcome by *prioritizing* the media units and using the priority information in transmission[1]. For example, MPEG reference frames can be assigned a higher priority than dependent frames. To be effective, the prioritization information must be used by the network layer.

CUDP is a best-effort network protocol that uses prioritization information. Experiments with CUDP show that smooth real-time video playback can be achieved even with a relatively fickle cross country Internet connection. For example, in an experiment where MPEG video was transmitted from U.C. Berkeley to Cornell University (2800 miles and 18 hops away), an average playback rate of over 17 frames per second was achieved and consecutive frame drops were rare. Of course, only a few such connections are possible due to the bandwidth limitations of today's Internet.

The rest of this paper is organized as follows. Section 2 describes algorithms to prioritize audio and video media units. Section 3 describes CUDP's error correction strategy, and section 4 describes CUDP's flow control algorithm. Section 5 briefly describes the results of experiments that characterize CUDP, and section 6 reviews related work and concludes the paper.

2. Prioritization of Audio and Video

This section describes a set of algorithms for prioritizing media units. More precisely, the problem that these algorithms attempt to solve is the following: given a set of n media units associated with a continuous media stream, assign an ordering to the units (called the *priority order*) such that, if a subset k of the units are received in priority order, the quality of the stream reconstructed using that subset is as high as possible. This section sketches two prioritization algorithms, one for uncompressed audio and one for motion-JPEG. MPEG prioritization algorithms are described in the author's dissertation [9], and beyond the scope of this paper.

1. A CM file server can prioritize the media units after reading them from disk but before handing them to the network layer.

In motion-JPEG video, each frame is compressed independently using JPEG compression [13]. Our prioritization algorithm for motion-JPEG attempts to minimize the chance of sequential frame losses by prioritizing media units using *inverse binary order* (*IBO*). The IBO of a sequence of media units is obtained by assigning each media units a *frame number*, starting from zero. The media units are then sorted using the reversal of the binary representation of the frame number as a key. The resulting sorted list is in IBO.

For example, suppose fifteen frames are prioritized using IBO. The list of frames in priority order, highest first, is: {0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7}. Suppose half the frames are lost due to congestion. Provided these are the lowest priority frames (a condition that is enforced by CUDP's error correction scheme), the resulting playback sequence will contain every other frame. IBO prioritization scatters lost frames evenly, so that the user perceives the effect of network loss as a lowering in the frame rate.

Uncompressed audio streams are composed of a set of samples taken at regular intervals. The rate of sampling, called the sampling rate, varies from 8 kHz for telephone quality audio to 44 kHz for CD quality audio. Because the individual samples are small (typically one to two bytes), many samples are usually sent in a single network packet.

Our prioritization algorithm begins by subsampling the audio signal. That is, a group of audio samples is subsampled into several packets, each of which are independently delivered to the destination. The receiver reconstructs an approximation of the original signal using the packets it receives and interpolating missing packets. Packets are prioritized using IBO to make the interpolation as accurate as possible.

For example, a media unit containing one second of 16 kHz audio (16,000 samples) is split into 4 packets, 4000 samples per packet. The first packet contains samples numbered $4i$ ($i \in [0,3999]$), the second packet contains samples numbered $4i + 1$, and so on up to the last packet. If the first packet is received, a low quality (4 kHz) audio signal is played. If two packets are received, an 8 kHz signal can be played. Thus, the effect of packet loss is changed to a reduction in signal quality.

3. Best-Effort Prioritized Delivery

Having discussed prioritization algorithms, we now turn to the problem of transmitting prioritized media units. Cyclic-UDP (CUDP) is a transmission protocol based on UDP that supports the delivery of prioritized media units. It uses an error correction strategy that makes the probability of the successful delivery of a media unit proportional to the unit's priority. This section describes CUDP's error correction mechanism to support this function.

At the beginning of the cycle, CUDP fragments the media units into *packets*, attaches a header to each packet (for sequencing and loss detection), and places the packets in a queue called the *packetQueue*. The packetQueue is sorted by packet priority: high priority packets are near the front of the queue, low priority packets towards the rear. In addition, the sender marks each packet as *unsent*.

Packets are sent in a series of bursts, where up to *N* bytes are sent. The parameter *N* limits the burstiness of the stream. In each burst, the queue is scanned front to back and each unsent packets encountered is sent as a UDP datagram and marked as *sent*. The sender also records the burst number (called the *burstId*).

After sending a burst, CUDP waits before sending the next burst. The waiting time, *T,* is set so that the transmitted bandwidth matches the estimated channel bandwidth, *estBW*:

$$T = numSent/estBW$$

where *numSent* is the number of bytes sent in the previous burst. The calculation of *estBW* is described in the next section.

When the destination detects a missing packet, it sends an *automatic repeat request (ARQ)* requesting retransmission of missing packets. The ARQ, which is sent as a UDP datagram, contains a list of {*burstId, packetId*} pairs that specify packets missing at the destination. *PacketId* identifies the lost packet, and *burstId* is copied from the packet that triggered the ARQ.

When the sender receives an ARQ, it updates the packetQueue by marking the packets listed in the ARQ as *unsent*. As such, they will be sent during the next burst.

Since each non-sequential packet the destination receives triggers an ARQ for all missing packets, redundant ARQs may be sent. The sender uses the *burstId* of the ARQ to detect (and ignore) such redundant requests.

A special packet, called the *end-of-cycle* (*EOC*) packet, is used to detect lost packets from the end of the packetQueue. EOC packets contain no media data and are identified by a bit in the header. An EOC packet is sent when all packets in the packetQueue have been marked *sent*, and thereafter once every 100 milliseconds. When the destination receives an EOC packet, it issues an ARQ for all packets missing in the current cycle. EOC packets are needed, for example, when the last packet in the packetQueue is lost.

Figure 1 shows a sample transmission sequence of five packets. The tables below the figure show the relevant information in the *packetQueue* at the times indicated. In the burst 1, packets one and two are sent;

accordingly, the *burstId* of each packet is set to one and the packets are marked as *sent* (time T_1). Suppose packet one is lost, but packet two arrives intact. The destination marks packet two as received and sends an ARQ for packet one with the *burstId* equal to 1 (denoted *<ARQ b1, p1>*). Meanwhile, the source sends packets three and four in burst 2 (T_2), but packet three is lost. Shortly thereafter (T_3), the source receives *<ARQ b1, p1>* and marks packet one as *unsent*. When the destination receives packet four, it issues an ARQ for packets one and three (*<ARQ b2, p[1,3]>*). During burst 3, packet one and packet five are marked as *sent* and transmitted, with the *burstId* of each set to three. The source then receives the second ARQ, but ignores the redundant ARQ for packet one since the *burstId* in the ARQ is 2, but packet one was resent during burst 3. Only the request for packet three is valid, so packet three is marked as *unsent*. It is sent in burst 4, along with an end-of-cycle (EOC) packet.

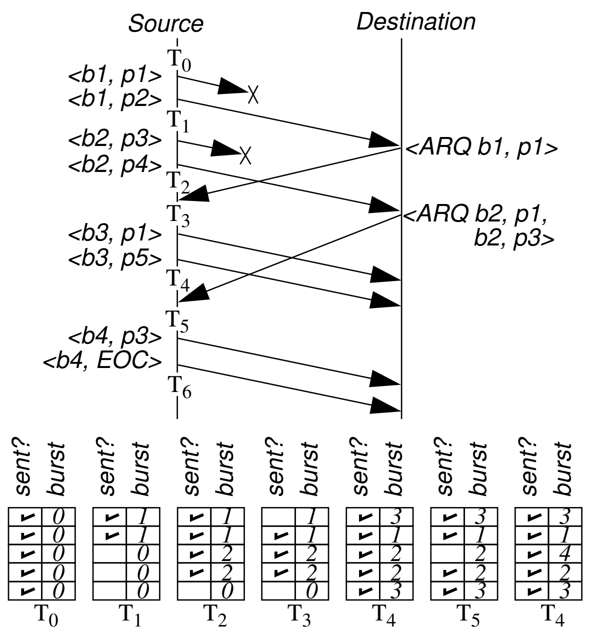

Figure 1: Cyclic-UDP example

CUDP gives high priority media units a better chance of delivery because, in the event of packet loss, they get more chances for retransmission than units later in the queue. Assuming each packet has probability $\alpha < 1$ of being lost, then the probability[2] of a packet being lost after k delivery attempts is α^k. Since k is approximately the cycle length divided by the round trip time, high priority packets can be given arbitrarily high probability of delivery by making the cycle lengths large (at the cost of extra buffering).

2. This calculation assumes that the chance of packet loss on each delivery attempt is independent.

4. Flow Control

CUDP uses an estimate of the available channel bandwidth, *estBW*, for flow control. The estimate is computed using measurements taken over periods known as *measurement periods* (*MPs*), whose boundaries are defined by the source. MPs typically last 100-200 milliseconds.

The measurement process during an MP can be broken into two phases. In the first phase, which takes place in the receiver, the following values are measured:

- *meanDelay*, the average end-to-end packet delay.
- *recvdBW*, the bandwidth received.
- *L*, the fraction of bytes lost.

To measure the end-to-end delay of each packet, the following method is used. When each packet is transmitted, a field in the packet header is set to the value on the sender's system clock. When the packet is received, this value is subtracted from the value on the receiver's system clock to get a difference, δ. The minimum value of δ over the MP is the skew of the system clocks plus the minimum flight time of a packet[3]. We call this quantity *S*. The value of *S* from the previous MP is subtracted from δ to get the measured delay:

$$M = \delta - S_{prev}$$

Finally, *meanDelay* is computed from *M* using the same method as TCP [10]:

$$err = M - meanDelay$$

$$meanDelay' = meanDelay + err/8$$

To compute *L* and *recvdBW*, the source counts the number of bytes transmitted (*bytesSent*) and the destination counts the number of bytes received (*bytesRecvd*). The counters are reset at the beginning of each MP. The source sets three fields in the header of every packet that indicate 1) the MP identifier (used by the destination to detect new MPs), 2) the number of bytes sent by the source during the previous MP, and 3) the duration of the previous MP (*mpDuration*).

When the receiver detects a new MP, it computes *L* and *recvdBW*:

$$L = 1 - bytesRecvd/bytesSent$$

$$recvdBw = bytesRecvd/mpDuration$$

Finally, the destination packages up the computed values of *meanDelay*, *recvdBW*, and *L* in a *feedback unit,* which is sent to the source as a UDP datagram, ending phase one.

3. A side effect of this measurement process is that the skew can be used to provide crude clock synchronization between the sender and the receiver.

In the second phase, which takes place at the source, the feedback unit is used to calculate two parameters, *estBW* and *ltBW*. *LtBW* is updated using a weighted average of the *recvdBW* measurements:

$$ltBW' = ltBW + (recvdBW - ltBW) / 64$$

LtBW ("long-term bandwidth") is an estimate of the maximum bandwidth available on the connection in the absence of congestion. As such, it is only updated when either the loss is non-zero, which means the connection is uncongested, or when *recvdBW* is greater than the current value of *ltBW*, which means the current value of *ltBW* is low.

Once *ltBW* is updated, *estBW* is computed using *ltBW*, *meanDelay*, *L*, and two parameters associated with the connection, the expected loss rate (X) and the expected delay (D_t). The following strategy is used:

```
if (L > X) {
    estBW = (1-L)*ltBW/(1-X);
} else if (meanDelay > Dt) {
    estBW = (1+X)*ltBW*Dt/meanDelay;
} else {
    estBW = (1+X)*ltBW;
}
```

The effect of this strategy is to lower estBW if either the loss or the delay is higher than expected (i.e., if $L>X$ or $meanDelay > D_t$). Otherwise, the bandwidth is set to the *ltBW* plus some extra bandwidth proportional to the expected loss rate. Finally, if the result is outside the range [*minBandwidth..maxBandwidth*], *estBw* is set to either *minBandwidth* or *maxBandwidth* to ensure that the bandwidth never rises above what is needed or sinks below a threshold.

One side-effect of this flow control scheme is that the system can control the relative bandwidth allocated to different streams. For example, suppose an audio and a video stream are sent from a file server to a client using independent CUDP connections. By setting the expected drop rate of the audio stream to be higher than that of the video stream, the audio stream will receive more bandwidth than the video stream when congestion occurs, since the video stream will lower its transmission rate first. Consequently, audio quality will be maintained through congestion, at the expense of video quality.

5. Experimental Results

This section describes a set of experiments performed to evaluate the performance and behavior of CUDP in various scenarios. The questions we wanted to answer were:

1. How well does CUDP estimate and share the bandwidth available on a network?
2. How well does it work on different networks (e.g., LAN or WAN)?
3. How well do the prioritization algorithms work?

Two canonical movies were used in these tests. The audio portion of both movies was encoded as 8kHz uncompressed audio. The video portion of one movie was encoded as a 352x240 MPEG bitstream ("Ferris Wheel"), and the video portion of the other movie was encoded as a 320x240 motion-JPEG bitstream ("Andre and Wally B."). Each movie lasted 60 seconds. Table 1 lists characterizes the video portion of these movies.

Movie	Bitrate (Mbits)	Frame-Size (KBytes) min/avg/max
Ferris	1.25	2.0/5.1/16.6
Andre	2.83	7.9/11.5/13.1

Table 1: Characterization of the experimental streams

Six experiments were conducted on three networks, for a total of 18 experiments. The networks used were a 10 Mbit/second Ethernet (LAN), a metropolitan area network with three subnets connected by two gateways (MAN), and an Internet connection between UC Berkeley and Cornell University with 18 gateways (WAN). We estimated the maximum usable bandwidth of each network by sending several unregulated streams of datagrams at varying rates and measuring the maximum end-to-end bandwidth. The results indicate that the LAN was capable of 9 Mbits/second, the MAN was capable of 5 Mbits/second, and of the WAN was capable of 1.2 Mbits/second. We also characterized the available bandwidth of all three networks using TCP. The results were 2.6 Mbits on the LAN, 580 Kbits on the MAN, and 104 Kbits on the WAN.

The experiments consisted of sending one to four simultaneous copies of a movie using CUDP on each network. Traces of the received packets were recorded and used to calculate the fidelity of the streams that would be played at the receiver. The fidelity of audio was measured as the effective sampling frequency of the reconstructed stream, while the fidelity of video was measured as the playback rate (in frames/second -- fps) and the *playback jitter*. Playback jitter is a measure of the smoothness of the video stream. It is defined as the standard deviation of the interframe playback time computed over a one second window. Small values of jitter correspond to smooth play, large values to jerky play. A rule of thumb is that the playback is perceived as smooth if the playback jitter is less than half the interframe playback time. For a more complete description of the experimental conditions, results, and a discussion of playback jitter, see [9].

Test	Mbits	LAN				MAN				WAN			
		FPS		Jitter		FPS		Jitter		FPS		Jitter	
		Min	Avg	Max	Avg	Min	Avg	Max	Avg	Min	Avg	Max	Avg
Ferris-1	1.25	30.0	30.0	0.0	0.0	30.0	30.0	0.0	0.0	4.0	17.4	194	28.6
Ferris-2	2.5	30.0	30.0	0.0	0.0	30.0	30.0	0.0	0.0	4.0	16.2	194	30.6
Ferris-4	5.0	30.0	30.0	0.0	0.0	6.0	28.6	112	4.2	2.0	7.7	613	116
Andre-1	2.83	30.0	30.0	0.0	0.0	20.7	25.5	17.0	11.5	5.0	9.0	118	32.4
Andre-2	5.66	25.0	29.6	13.6	1.7	17.0	24.0	17.1	13.4	5.0	8.4	122	35.2
Andre-4	11.2	8.0	20.2	44.1	15.3	2.0	12.1	518	37.0	2.0	4.4	660	121

Table 2: Experimental Results

Table 2 shows the measured fidelity of the reconstructed video. Each experiment is listed by the network environment, the movie, and the number of simultaneous streams. For example, the row labeled "Ferris-4" corresponds to the experiments using 4 simultaneous MPEG streams. The average values are computed by averaging over all four streams; the worst-case value are computed using the worst-case of any of the streams.

In most cases, the fidelity of the Ferris sequence was quite good. Note, for example, that it played perfectly on the LAN, and very well on the MAN. On the WAN, the average frame rate was over 17 frames per second (fps) for one stream, 16 fps for two, and 7 fps for four. Moreover, the playback was smooth on average, as indicated by the low playback jitter. The frame rate on the WAN, however, occasionally dropped to unacceptable quality (2 to 4 fps).

Andre also played well up to the point where the net was overloaded (e.g., trying to send 11 Mbits on the LAN). Moreover, the average jitter was low, indicating that the prioritization helped smooth out the network congestion.

The fidelities of the reconstructed audio streams are not listed due to space, but the quality was typically quite high. The effective sampling rate was 6kHz in the worst cycle, and 7.9 kHz on average.

6. Related Work and Conclusions

Many other researchers have investigated best-effort transmission protocols. TCP [10] is perhaps the most well known best-effort protocol, and the Xphone system at Columbia uses TCP for audio and video transmission [3]. Most other systems are based on UDP. Jeffay [5,11,12] has investigated several techniques to detect congestion quickly and to use this information to adapt transmission parameters. Their work is intended for low latency applications like video conferencing. Nv [4], a video conferencing tool popular on the Internet, uses a custom compression technique that is robust to packet loss. Delgrossi and Halstrick have proposed using multiple multicasting trees to support best-effort delivery of prioritized media units[2]. Yavatkar and Manoj study two forward error correction (FEC) strategies in a multicasting simulation study [14]. And, the Priority Encoded Transmission project [1], implements an FEC strategy that uses prioritization. Their method has the property that it requires no flow control or ARQs and high priority media units can be reconstructed even in high loss conditions.

The primary contribution of this work is 1) it details encoding-specific prioritization algorithms that maximize the quality of the reconstructed stream and 2) it describes a ARQ transmission protocol that delivers media units in priority order. Our future work will include extending this work to support lower latency applications such as video conferencing.

References

[1] A. Albanese, J. Blomer, J. Edmonds, M. Luby, M. Sudan, *Priority Encoding Transmission*, Symposium on Foundations of Computer Science, October, 1994.

[2] L. Delgrossi, C. Halstrick, et. al., *Media Scaling for Audio Vidual Communication with the Heidelberg Transport System*, Proceedings of ACM Multimedia 93, (Anaheim, CA, August 1-6, 1993). Association for Computing Machinery Press, New York, 1993 pp. 99-104

[3] A. Eleftheridas, S. Pejhan, D. Anastassiou, *Algorithms and Performance Evaluation of the XPhone Multimedia Communication System*, Proceedings of ACM Multimedia 93, (Anaheim, CA, August 1-6, 1993). Association for Computing Machinery Press, New York, 1993 pp. 311-320

[4] R. Frederick, *Experiences with Real-time Software Video Compression*, Proceedings of the Packet Video Workshop (Portland, Oregon, USA, September 26-27, 1994)

[5] K. Jeffay, D. L. Stone, T. Talley, and F. D. Smith, *Adaptive, best-effort delivery of digital audio and video across packet switched networks*, Network and Operating System Support for Digital Audio and Video: Third International Workshop, (La Jolla, California, USA, November 12-13, 1992), Springer-Verlag, Berlin; New York, pp. 1-12,

[6] S. G. Kochan et. al, *UNIX Networking*, Hayden Books, Carmel, IN, 1989

[7] D. Le Gall, *MPEG: a video compression standard for multimedia applications,* Communications of the ACM, April 1991, Vol. 34, Num.4, pp. 46-58.

[8] K. K. Ramakrishnan, et. al., "Operating System Support for a Video on Demand File Service," Network and Operating System Support for Digital Audio and Video: Fourth International Workshop, Lancaster, UK November 1993. Springer-Verlag, Berlin; New York pp 225-236

[9] B. C. Smith, *Implementation Techniques for Continuous Media Systems and Applications*, Ph.D. Dissertation, University of California, Berkeley, September 1994.

[10] W. Stevens, *Tcp/Ip Illustrated*, Addison-wesley, New York, 1994.

[11] D. Stone, K. Jeffay, *Queue Monitoring: A Delay Jitter Management Policy*, Network and Operating System Support for Digital Audio and Video: Fourth International Workshop, Lancaster, UK November,1993. Springer-Verlag, Berlin; New York pp. 151-162

[12] T. Talley, K. Jeffay, *Two-Dimensional Scaling Techniques for Adaptive, Rate-Based Transmission Control of Live Audio and Video Streams*, Proceedings ACM Multimedia '94, San Francisco, California, Oct 1994

[13] F. Tobagi, J. Pang, *Starworks *TM -- A Video Applications Server*, Proceedings of IEEE Compcon '93 (San Francisco, California, USA, February, 1993).

[14] G. K. Wallace, *The JPEG Still Picture Compression Standard*, Communications of the ACM, Vol. 34, Num. 4, pp. 30-44, April 1991.

[15] R. Yavatkar, L. Manoj, *Optimistic Strategies for Large Scale Dissemination of Multimedia Information*, Proceedings of ACM Multimedia 93, (Anaheim, CA, August 1-6, 1993). Association for Computing Machinery Press, New York, 1993, pp. 13-20