CORBA Evaluation of Video Streaming wrt QoS Provisioning *

Wonjun Lee and Jaideep Srivastava
Department of Computer Science & Engineering
University of Minnesota, Minneapolis, MN 55455
{WJLEE|SRIVASTA}@CS.UMN.EDU

Abstract

This paper describes *the design,* implementation, and evaluation of CORBA and Socket-based Continuous Media *(CM) systems. TCP/IP* is not suitable for distributed applications which require high network bandwidth and timing-criticality. *UDP/IP* is one of *the* alternatives. *However,* due *to the* fact that *UDP* is a *lossy* protocol, many issues arise *when* implementing distributed *CM* applications. *Most of the QoS* (Quality *of* Service) metrics known so far assume that *the* communication channel is lossless. *In this* paper, since we use *UDP for CM data* transmission, we adopt a new *QoS* metric *that is* applicable *to lossy* streams *to* evaluate *the* performance *of* our *CM* server. To reduce *QoS Loss* factors and *Drift* factors, we adopt a new *strategy, called* QoS-Driven Dropping *Mechanism, for the CM* server. *Besides the* traditional *C-socket (TCP- UDP/IP)-based CM* server mechanisms, we implemented our *CM* server on *CORBA. It* turns *out that the CORBA-based implementation* run considerably slower *than the UDP-version, but faster than the TCP* version.

1 introduction

Continuous media (CM) servers have recently been a hot research topic for several reasons. Network speed has been continuously increasing, and therefore services like *Video on Demand, Teleconferencing,* Distance *Learning, etc.* are likely to be popular in everyday life. However, given the limitations of current network bandwidth, straightforward TCP implementations are not suitable for such bandwidth-sensitive applications. TCP has its own flow control mechanisms, error detection and retransmissions, all of which add extra time as well as network bandwidth overhead to the transmission. This causes unexpected and unpredictable delay and jitter when transferring CM data, while timing is one of the most critical requirements of CM applications. However, many CM applications don't need highly reliable transmission. Losing some frames is less important than having too much delay jitter or losing synchrony between streams. Thus we conclude that UDP is more suitable for CM applications than TCP. Especially, we can take advantage of the fact that even though UDP is a lossy protocol, the number of frames lost is not that many, and in most cases, is still in an acceptable range. This fact gives us an important observation that adding some control channel, having more intelligent dropping mechanisms will give us good performance in terms of both timing quality and number of lost frames.

We ported our original socket(TCP-UDP/IP)-based CM server [2] to CORBA. We use version 2.0 of Orbix from IONA Technologies as the ORB [1]. This replaces all C socket calls with stubs and skeletons generated from a pair of CORBA interface definition language (IDL) specifications. The IDL specification uses *sequence* parameters for the data buffer rather than *string* parameters which are slower. Due to the higher fixed overhead of CORBA such as demultiplexing and memory management, this version shows much lower performance.

In this paper, we present the design, implementation and performance evaluation of a QoS-driven CM server based on CORBA and Sockets. In section 2,

*This work is supported by Air Force contract number F30602-96-C-0130 to Honeywell Inc, via subcontract number B09030541/AF to the University of Minnesota.

we describe our motivation and objectives in designing our CM server and QoS metrics we used in the experiment. Section 3 presents the detailed design of our CM server including the architecture, QoS-driven dropping mechanism and implementation issues. In section 4, we present the results of an experimental evaluation of our CM Server system based on the QoS metrics described in section 2 for several variable factors such as file systems, network protocols, server types, and number of streams. Section 5 presents concluding remarks.

2 Motivation & Objectives

In order to achieve high performance from CM servers, the design must consider the resource constraints as well as the properties of CM streams. CM streams have their own features and special *QoS* metrics. Since we opted to design our CM server and clients based on the lossy UDP protocol, we should measure its performance using the appropriate QoS metrics. These QoS metrics play an important role in our &OS-driven CM server, in particular the QoS Manager.

2.1 QoS Metrics

Wijesekera [5] defined a set of metrics that are suitable for measuring the lossy nature of UDP based CM communication. Continuity of a CM stream is metrized by three components; namely *rate*, *drift* and *content*. For the purposes of describing these metrics, we envision a CM stream as a flow of data units (referred to as logical data *units -LDU's* in the uniform framework of [4]). The ideal rate of a flow and the maximum permissible deviation from it constitute our rate parameters. Given the ideal rate and the beginning time of a CM stream, there is an ideal time for a given LDU to arrive at the client, e.g. to be displayed. Given the envisioned fluid-like nature of CM streams, the appearance time of a given LDU may deviate from this ideal. The rate variations can be measured more accurately by drift parameters. Our *drift* parameters specify aggregate and consecutive non-zero drifts from these ideals, over a given number of consecutive LDU's in a stream.

In addition to timing and rate, ideal contents of a CM stream are specified by the ideal contents of each LDU. Due to loss, delivery, or resource overload problems, appearance of LDU's may deviate from this ideal, and consequently lead to perceived discontinuity of CM streams. Our metrics of continuity are designed to measure the average and bursty deviation from the ideal specification. A loss or repetition of a LDU is considered a unit loss in a CM stream. (A more precise definition is given in [5].) The aggregate number of such unit losses is the aggregate loss of a CM stream, while the largest consecutive non-zero loss is its *consecutive loss*.

Human response to video and audio is quite interesting. According to [5], up to 23% of aggregate video loss and 21% of aggregate audio loss are tolerable. The acceptable values for consecutive loss of both video and audio are approximately 2 LDU. Up to about 20% of video and 7% of audio rate variations are tolerable.

2.2 Objectives

It is a reasonable requirement to expect a CM server to guarantee that all the QoS parameters defined above are met. When the load on the CM server is low, it is possible to meet this requirement. However, when the number of concurrent CM streams increases in the CM server, it becomes difficult to guarantee all the QoS parameters. It is easy to guarantee only ALF and CLF by delaying the following LDUs, which makes the ADF and CDF unacceptable. On the other hand, it is easy to guarantee only ADF and CDF by delaying the early LDUs and dropping the late LDUs, which makes the ALF and CLF unacceptable. Given certain resources, we want to support as many CM streams as possible, whose QoS parameters are all within acceptable limits. There are three possible approaches: (i) to guarantee ALF and CLF first, (ii) to guarantee ADF and CDF first, and (iii) to compromise between guaranteeing ALF/CLF and guaranteeing ADF/CDF. Furthermore, as more clients require CM streams, the quality of service of CM server will degrade. It is desired that the degradation be graceful. To guarantee each client to be served with some reasonable quality, the admission control is also necessary.

3 Design of CM Server

In this section, we describe the architecture and design of our CM server. An important aspect is its QoS-driven dropping mechanism.

3.1 Architecture

The CM server has a typical *client/server* architecture. It includes one CM server and one or more CM clients, which are served concurrently. Figure 1 shows the architecture where the server provides multiple streams to the requesting clients across the network.

The CM server has four types of components, i.e. single instantiations of the *Network* Manager and the *QoS Manager,* and multiple instantiations of the *Proxy Server* and the *I/O Manager.* There are as many proxy servers as clients. The *Network* Manager responds to clients' connection requests. The *QoS Manager* is responsible for admission control and I/O scheduling. Each *Proxy Server* communicates with a client, receiving CM stream operation requests and sending CM data across the network. Each *I/O Manager* reads CM data from disks on behalf of the corresponding proxy server. The CM client is relatively simple compared with the CM server, and has two main components, namely a *Client N/W* Controller and a *CM Player.*

3.2 QoS Driven Dropping Mechanism

The major task of the proxy server is to send the CM stream to the client, and ensure the desired data rate and QoS requirements. The proxy server divides its service time into service cycles. The length of a service cycle is decided by the playback rate of the CM stream. For instance, if the data rate is 30 frames/second, the service cycle is 1/30 second long. In the beginning of each cycle, the proxy server wakes up and sends out a LDU. Then it waits till the beginning of the next service cycle. Once the CM stream begins, every service cycle is associated with a LDU. A LDU is late for a service cycle if it is not ready at the beginning of the service cycle. In general cases, the proxy server wakes up on time and sends out the next LDU. However, there are some exceptions. When a service cycle begins, the related LDU may not be ready for potentially two reasons. First is that the LDU is scheduled not to be read from the disk at all. Second is that the proxy server may wake up late in a service cycle because the non-real-time operating system can not guarantee the required timing. We allow a small interval of time t according to the permissible drift. If the proxy server does not wake up till t has elapsed in a service cycle, the proxy server is considered late.

We propose three approaches to handling these problems:

- The first approach is to send the LDUs sequentially without any LDU's being dropped. This approach, which we call the *sequential mechanism,* favors the ALF and CLF QoS parameters. Although the sequential mechanism has the best result for ALF and CLF, performance on other QoS parameters may be very bad and the system's capability is restricted.
- The second approach is called the *pure* dropping *mechanism.* When a LDU is late or the proxy server wakes up late, the proxy server drops the LDU and sends the next LDU instead. The pure dropping mechanism favors the ADF and CDF QoS parameters. The drift factors gets the best results, but the LDU loss may increase to an unacceptable level.
- The third approach tries to compromise between the loss factors and the drift factors, and is called *QoS driven dropping* mechanism. It is done by mainly keeping the CLF less than 3. In this way, a LDU is dropped only when the dropping doesn't affect the video or audio quality, and the drift factors are kept as low as possible. Furthermore, in high load some LDUs are not retrieved from the disks to save I/O bandwidth. The proxy server also knows which LDUs are not retrieved. So, this QoS driven dropping mechanism helps to provide good performance with graceful degradation.

3.3 Integration with CORBA

Extending the socket interface to use CORBA requires some modifications to the original C/Socket code. We replaced all C socket calls with stubs and skeletons generated from a pair of CORBA interface definitions. One IDL interface (called *CM-User)* uses a sequence to transmit the data from server to client, and the other IDL interface (called *CM-Request)* has operations for opening a video stream from the server and six video functions such as play, fast-forward, slow-forward, pause, resume and stop. The video functions change the rate of playout with the client's process id and a given play rate. The *putMJPGFrame* operation of *CM_User,* which is part of the client interface is called from the server (in proxy server) with two parameters: a sequence of MJPEG frames and

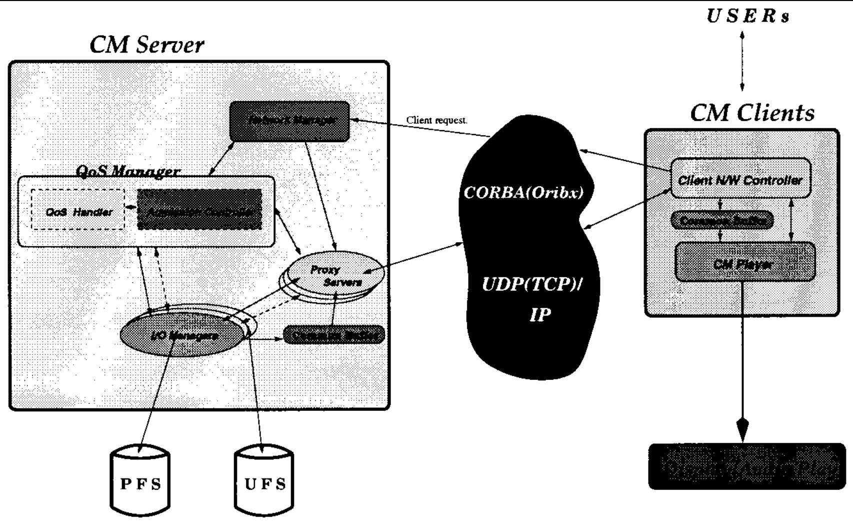


Figure 1: CM Server-Clients Architecture

its length. The *playMJPG* operations of ***CM-Request*** in the server side use *oneway* semantics for video functions since video distribution needs only unidirectional control signal transfer, i.e. from client to server.

We ported the first version of socket-based CM server to CORBA. The CORBA implementation was developed using a single threaded version of Orbix 2.0 which fully supports the OMG 2.0 CORBA standard [1]. All C/socket calls were replaced with stubs and skeletons generated from a pair of CORBA interface definition language (IDL) specifications. The IDL specification uses ***sequence*** parameters for the data buffer rather than ***string*** parameters, which are somewhat slow. It is because the IDL ***sequence*** mapping contains a length field, whereas the ***string*** mapping does not. This length field helps the IDL stub easily determine the end of the sequence.

The main drawback of using CORBA in CM Server is that the data copying overhead and the higher fixed overhead of CORBA considerably limits its performance. For small buffer sizes, the higher fixed overheads of CORBA, such as memory management, makes the performance lower. For large buffer sizes, another factor such data copying overhead significantly affects the performance, and limits the throughput. Every time a request is made to the CM server, the request message of CORBA contains the name of its intended remote operation represented as a string. Thus, CORBA demultiplexes incoming request messages to the appropriate upcall by performing a linear search through the list of operations in the IDL interface. Henceforth, operations in CORBA-based CM server should be ordered by considering this, i.e. decreasing frequency of use.

4 Experimental Evaluation

In this section, we compare the performance results of the three kinds of CM server architectures (TCP/UDP/CORBA)-based on experiments which use QoS metrics such as ALF, CLF, ADF and CDF.

4.1 Evaluation Metrics

In table 1, we show the metrics used in our experimental evaluation. We measured the performance of CM servers by having servers run on *rawana.cs.umn.edu* and having CM clients run on another machine (*sejong.cs.umn.edu*) in the Computer Science Department at the University of Minnesota. Two different file systems were used: (1) PFS (Presto File System) developed at the University of Minnesota [3] (2) UFS (the most conventional file system, i.e. Unix file system). The QoS metrics we measured are ALF, CLF, ADF, and CDF, which were all described in detail in section 2.1.

4.2 Experimental Results

The main capability of the CM Server is its *QoS-**driven dropping*** mechanism based on the network

Testing Sites	CM Server types	QoS Metrics	# of Streams	File Systems
sejong.cs.umn.edu **(CS Dept, U of MN)**	**CM-Server1 (TCP)** **CM-Server2 (UDP)** **CM-Server3 (CORBA)**	**ALF** **CLF** **ADF** **CDF**	1 ... **12**	**PFS** **UFS**

Table 1: Metrics Used in the Experiments

traffic, to uniformly maintain the QoS drift factors such as ADF and CDF. The dropping mechanism in CM Server drastically reduces time factors under heavy network traffic. Yet, we should be able to consider the bad effects gained by lost of lost frames under the situation of many streams running. We tried to exclude the other undesired factors issued by other processes running during our experiments.

We also did experiments using CORBA-Based CM Server system and compared its performance results with those of Socket (TCP-UDP/IP)-Based CM Server systems, and figured out that the performance of CORBA-Based system is better than that of TCP/IP-Based system, but it is worse than that of UDP/IP-Based CM Server system on the five QoS metrics. We used the only Presto File System in this experiment. (Figure 2(a), 2(b), 2(c), 2(d)).

5 Conclusions

In this paper, we described a CM server architecture and techniques to implement distributed CM applications. We have described QoS metrics that are applicable to a lossy channel like UDP, and how we evaluated the performance of our CM server based on these metrics. We described the design issues and implementation details of CM servers using both C/sockets and CORBA as distributed communication mechanisms. The QoS metrics include specification of retrieval rate and its variation, allowable timing drift, acceptable average and bursty errors, and degree of synchronization. Our on-going work includes developing intelligent algorithms and mechanisms of admission control and QoS management that take advantage of QoS specifications to optimize system performance.

References

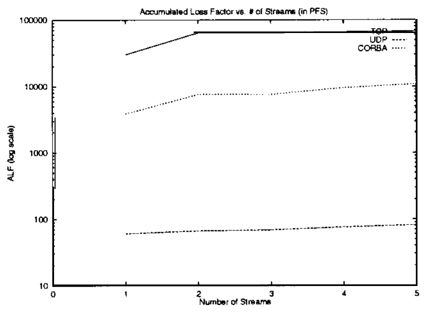
[1] *Orbix 2* Programming Guide . IONA Technologies, 1995.

[2] Wonjun Lee, Difu Su, and Jaideep Srivastava. A QoS-Driven Networked Continuous Media Server. In *Proceedings of SPIE International Symposium on* Lasers, *Optpelectronics, and Microphonics (Electronic* Imaging *and Multimedia Systems: SPIE Symposium on Photonics* China *- PC'98)*, Beijing, China, September 1998.

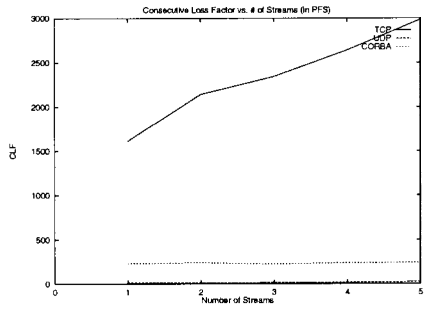
[3] Wonjun Lee, Difu Su, Jaideep Srivastava, D.R. Kenchammana-hosekote, and Duminda Wijesekera. Experimental Evaluation of PFS Continuous Media File System. In *6th ACM Int'l Conf. on Information and Knowledge Management (CIKM '97)*, November 1997.

[4] Ralf Steinmetz and Gerold Blakowski. A Media Synchronization Survey: Reference Model, Specification and Case Studies. *IEEE Journal on Selected* Areas in *Communication,* 14(1):5–35, 1996.

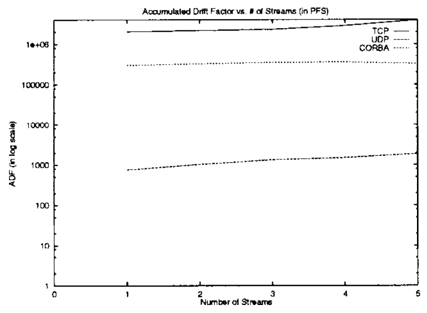
[5] Duminda Wijesekera and J. Srivastava. Experimental Evaluation of Loss Perception in Continuous Media. 1998. ACM-Springer Multimedia Systems Journal.



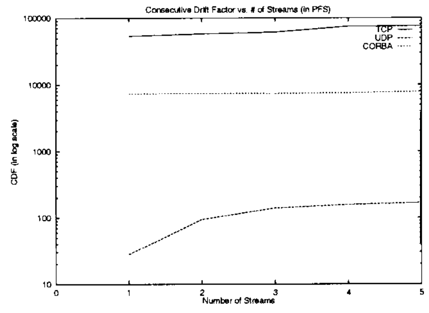
(a) ALF



(b) CLF



(c) ADF



(d) CDF

Figure 2: Experimental Results on QoS Metrics for Socket- vs. CORBA-Based CM Server Systems