P2P Video Synchronization in a Collaborative Virtual Environment

Suhit Gupta and Gail Kaiser

Columbia University, 500 W. 120th Street,
New York, NY 10027, United States
{suhit, kaiser}@cs.columbia.edu

Abstract. We previously developed a collaborative virtual environment (CVE) for small-group virtual classrooms, intended for distance learning by geographically dispersed students. The CVE employs a P2P approach to the frequent real-time updates to the 3D virtual worlds required by avatar movements (fellow students in the same room). This paper focuses on our extensions to support group viewing of lecture videos, called VECTORS, for Video Enhanced Collaboration for Team Oriented Remote Synchronization. VECTORS supports *synchronized* viewing of lecture videos, so the students all see "the same thing at the same time", and can pause, rewind, etc. in synchrony while discussing the lecture via "chat". We are particularly concerned with the needs of the technologically disenfranchised, e.g., whose only Internet access if via dialup networking. Thus VECTORS employs semantically compressed videos with meager bandwidth requirements.

1 Introduction

Learning is essentially a social activity and is of paramount importance in engineering project-based courses, where a high degree of cooperation is required [8]. The Columbia Hypermedia IMmersion Environment (CHIME) system [5] [6], created by the Programming Systems Lab (PSL – http://www.psl.cs.columbia.edu) at Columbia University, was designed as a framework for distributed software development environments. CHIME's users would be software project team members who might be geographically dispersed, but could be *virtually* collocated within the same "room" or adjoining "rooms" of a MUD-like 3D virtual world. The layout and contents of this *groupspace* represent the software project artifacts and/or the on-going software process. This model is similar to the one at MIT iLabs [14].

CHIME has more recently evolved into a general collaborative and information management infrastructure. One example of the utilization of CHIME's framework architecture is the visualizing of segments of videos that are pre-taped lectures of classes held here in the Computer Science Department at Columbia University. Distance learning programs such as the Columbia Video Network and the Stanford Center for Professional Development have evolved from mailing (via Fedex and the like) lecture video tapes to their off-campus students to streaming the videos over the

Internet. The lectures might be delivered “live”, but are frequently post-processed and packaged for students to watch (and re-watch) at their convenience. This introduces the possibility of forming “study groups" among off-campus students who view the lecture videos together, and pause the video for discussion when desired, thus approximating the pedagogically valuable discussions of on-campus students. Although the instructor is probably not available for these discussions, this may be an advantage, since on-campus students are rarely afforded the opportunity to pause, rewind and fast-forward their instructors' lectures.

However, collaborative video viewing by multiple geographically dispersed users is not yet supported by conventional Internet-video technology. It is particularly challenging to support WISIWYS (what I see is what you see) when some of the users are relatively disadvantaged with respect to bandwidth (e.g., dial-up modems) and local computer resources (e.g., archaic graphics cards, small disks). The VECTORS (Video Enhanced Collaboration for Team Oriented Remote Synchronization) plug-in was added to CHIME to allow users to synchronize on video based data. This was done by combining techniques that extract key frames from a video stream to create a semantically rich version of the video [13] and fast peer-to-peer UDP packet based synchronization [7], we allow groups of users to watch videos in synchrony, regardless of their bandwidth limitations. We have adopted technology (developed by others, Liu and Kender [13]) for “semantically compressing" standard MPEG videos into sequences of still JPEG images and utilized P2P techniques for synchronizing the semantic content across various clients.

2 Related Work

There has been a rich amount of work done in the field of Collaborative Virtual Environments (CVE) over the years. The key feature of research in CVE has been the social engineering aspect and the attempt to improve the user interface over which users communicate seamlessly with others [8] [15]. Prasolova-Forland discusses the mechanisms employed to improve social awareness in education [8][9] and has found that the traditional technical tools are not enough, and the mechanisms offered by CVEs provide a more promising supplement to the mechanisms in use already.

The advantage the 3D CVEs, with a MUD like interface, gives over traditional web-based collaborative environments is the ability for users to see what his/her peers are doing. We discuss CVEs further in our paper further describing CHIME. [28] In addition to the work that has gone into virtual environments that are geared towards educational purposes, stream synchronization is a widely studied topic in multimedia.

Most intra-stream synchronization schemes are based on data buffering at the sink(s) and on the introduction of a delay before the play-out of buffered data packets (i.e., frames). Those synchronization schemes can be rigid or adaptive [26]. In rigid schemes, such as [22], the play-out delay is chosen a priori in such a way that it accounts for the maximum network transfer delay that can likely occur across the sinks. Rigid schemes work under a worst-case scenario assumption and accept the introduction of delays that may be longer than necessary, in order to maximize the synchronization guarantees they can over even in demanding situations.

Contrary to a rigid approach, adaptive schemes [17] [23] [24] re-compute the delay parameter continuously while streaming: they try to “guess” the minimum delay that can be introduced, which still ensuring synchronization under actual operating conditions. In order to enhance quality of service in terms of minimized play-out delay, those schemes must accept some temporary synchronization inconsistencies and/or some data loss, in case the computed delay results are at times insufficient (due, to variations in network conditions) and may need to be corrected on the fly.

Our approach to synchronization can be classified as a centralized adaptive scheme that employs a local clock and operates in a reactive way. The most significant difference compared to other approaches, such as the Adaptive Synchronization Protocol [17], the work of Gonzalez et al. [21], or that of Liu et al. [20] (which can all be used equally for inter- and intra-stream applications), is that our approach is not based on the idea of play-out delay. Instead, we take advantage of layered semantic compression coupled with buffering to “buy more time" for clients that might not otherwise be able to remain in sync, by putting them on a less demanding level of the compression hierarchy.

Liu et al. provide a comprehensive summary of the mechanisms used in video multicast for quality and fairness adaptation as well as network and coding requirements [19]. To frame our work in that context, our current design and implementation models a single-rate server adaptation scheme to each of the clients because the video quality we provide is tailored specifically to that client's network resources. The focus in our work is directed towards the client-side end-user perceived quality and synchrony, so we did not utilize the most efficient server model. The authors believe that it would be trivial to substitute in a simulcast server adaptation model [26]. Our design also fits into the category of layered adaptation. Such an adaptation model defines a base quality level that users must achieve. Once users have acquired that level, the algorithm attempts to incrementally acquire more frames to present a higher quality video. In the work presented here, the definition of quality translates to a higher frame rate.

With respect to the software architecture, our approach most resembles the Lancaster Orchestration Service [26], since it is based on a central controller that coordinates the behavior of remote controlled units placed within the clients via appropriate directives (i.e., the VECTORS video buffer and manager). The Lancaster approach employs the adaptive delay-based scheme described above; hence the playback of video focuses on adapting to the lowest bandwidth client. That approach would degrade the playback experience of the other participants to accommodate the lowest bandwidth client. Our approach seems preferable, since it enables each client to receive video quality commensurate with its bandwidth resources.

Cen et al. provide a distributed real-time MPEG player that uses a software feedback loop between a single server and a single client to adjust frame rates [4]. Their architecture incorporates feedback logic within each video player and does not support synchronization across a group of players, while the work presented here explicitly supports the synchronization of semantically equivalent video frames across a small group of clients.

3 Our Solution

The goal was two-fold – to create a robust and dynamic collaborative virtual environment that would be a good enough framework for future plug-ins like video synchronization; and to create a near real-time video synchronization plug-in that would allow for students to participate in group based projects despite not being co-located.

3.1 CHIME

Our solution employs multiple extensible techniques that incorporate the advantages of the previous work on collaborative virtual environments. CHIME [5] [6] [28] is a metadata based information management and visualization environment, created to serve as a homogenous environment for heterogeneous applications and data for internet and intranet-based distributed software development. User movement however was the most interesting aspect with respect to the VECTORS plugin as it employed a P2P model. Since user position synchronization is a high frequency process, the publish/subscribe event system did not make for a good vehicle for this job, especially since the event system would add a large parsing overhead to each event that was as simple as coordinates in 3-space. We therefore do user synchronization using UDP packets on a peer-to-peer basis.

3.2 VECTORS

One of our goals for CHIME was to integrate video synchronization for users. Columbia University offers taped courses over the internet as part of their Columbia Video Network (CVN) department. These courses work well when the class is simply lecture based geared towards individuals with assignments that do not require group work. However, for courses like Software Engineering and Operating Systems, where team based software development is one of the critical pedagogical requirements, CVN is unable to deliver a full experience, especially since the students registered for these courses are geographically dispersed. Teams of students may need to watch multiple class lectures together and collaborate on them as they are in progress.

Students are not required by CVN to have the same resources in terms of bandwidth. In order to facilitate synchronized video feeds to diverse users, we had to deliver pre-canned and pre-processed semantically structured videos over heterogeneous Internet links to heterogeneous platforms in an efficient and adaptive manner. Video thus becomes an additional legitimate resource for mutual exploration in a distributed team's workflow.

Liu et al. [12] describe a similar project, however they are simply concerned with the QoS of the video and therefore their approach involves compression techniques

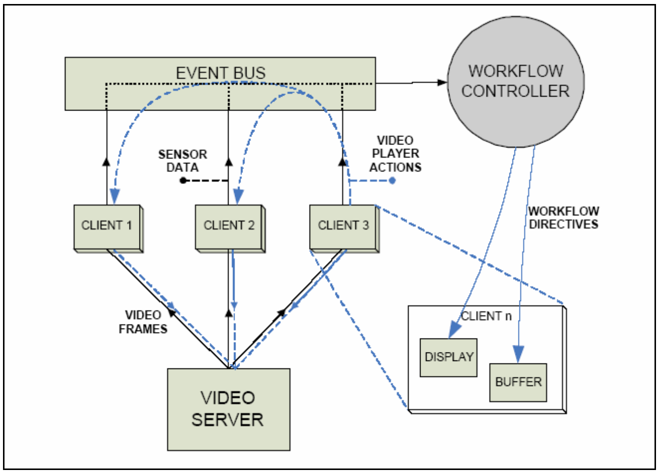

Fig. 1. The VECTORS Workflow

working with Mpeg-7 video. Moreover, they do not have the added requirement of embedding their video stream in a CVE. Our approach involves semantic structuring of the video, using technology previously developed by Liu and Kender [13]. Given this rich video stream consisting of the most representative frames, in terms of content, of the video, our goal was to try and give each user the best possible set of frames in order to enhance the video watching experience as much as possible while staying synchronized. However, instead of following approaches like those employed in commercial multimedia applications like Real Player (http://www.real.com/) or QuickTime (http://www.quicktime.com/) that drop every nth frame upon encountering network lag, which may have the negative side-effect of dropping important segments of the video, we procure separate levels of key frame density, each targeted at different bandwidth levels.

We still, however, have to give each client the correct video feed. In order to do this, our approach was four fold

1. Pre-fetch as many of the key frames as possible at the highest possible quality to the client before a pre-determined meeting time for the group. Meeting times can be ascertained by probing the user's schedule or by simply getting this information from the student directly. Though, it turns out that most videos are watched impromptu without any prior notice.
2. Probe the clients' bandwidth and number of cached frames and report results to the system periodically.
3. React to bandwidth changes in real time by lowering/raising the client to a lower or higher quality feed.
4. Allow pause, rewind, etc. in synchrony while discussing the lecture material via "chat".

All the video streams are made available by the video server. Probing is done by using software probes [10] [11], and reports of any changes are sent to the respective clients. Each client receives data and based on how much video it has in cache, its current position in the video and its bandwidth, the client determines what the highest quality frame it can download next successfully before it has to view it; and downloads it. This will continue until the end of the video.

3.2.1 The Server

VECTORS was proposed to analyze automatic methods for deriving semantic video structure, by finding large-scale temporal and spatial patterns, by detecting redundancies and semantic cross-correlations over long disjoint time intervals, and by compressing, indexing, and highlighting video segments based on semantically tagged visual sequences. We further explored user interaction in distributed environments in both a three-dimensional virtual world as well as a local two-dimensional client. We also analyzed various server cluster configurations, wire protocols, proxies, local client caches, and video management schemes.

The pre-canned and pre-tagged semantically structured video (Figure 2), was placed on the video server. Since the server simply provided the frames to each of the clients, the decision-making responsibility regarding synchronization fell upon the clients themselves; thus leading to a non-centralized decision-making system. The ultimate goal of the server was to analyze classes of particular server cluster configurations, wire protocols, proxies, local client caches, and video management schemes; however, in experiments, we simply treated the server as a black-box that would provide frames over an HTTP stream upon demand from a client. Example of a video frame hierarchy is shown in Figure 2, where we see two example levels of the same video stream. Level 1 has a sparse set of frames while Level 2 is denser, even

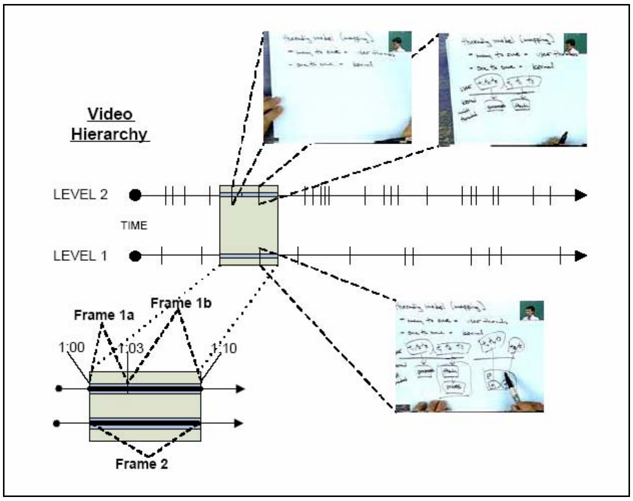

Fig. 2. Video Frame Hierarchy

though they semantically and pedagogically contain the same content. We would like to reiterate that audio was not semantically compressed and was therefore available as a separate and single file for the clients to download and play synchronously with the video stream.

Ultimately, the server consisted of two components, the semantically structured videos provided by Liu and Kender, and the scalable, proxy based video server. Since our goals lay in measuring the effectiveness the video synchronization in the 3D virtual client, we set up a simple web-server that contained the structured video content and simply served it to the clients.

3.2.2 The Client

The VECTORS Client Application, at the initial stage of development, focused on implementing, or at least making significant efforts to implement several functionalities which serve as the core of the VECTORS client side technology. The client that we chose for video synchronization was the CHIME client as it provided the perfect pluggable framework that allows users to see each other in a collaborative world where they can interact with one another and objects that represent heterogeneous back end data sources. The CHIME client is an authoring tool and perfect for pedagogical environments.

Since CHIME had the ability to visualize heterogeneous data sources and was built as a framework, VECTORS was built as a plug-in that visualized video with the added component that synchronized the video. Some of the basic components added to the VECTORS plug-in are –

GroupWare Synchronization – It provides a group-wide viewing session of a given video, each client remaining in sync with an overall video timeline. This is accomplished even if the various clients are at different network speeds (And thus are downloading a variety of different frames from the structured hierarchy that exists on the server).

Video Player in CHIME's 3D Environment – The player is designed to work inside the existing 3D environment offered by CHIME. CHIME utilizes a Crystal Space graphics engine, and all aspects of the video player must comply with constraints set forth by Crystal Space to ensure error free, 3D video display.

Downloadable Video Over HTTP – The video components after being processed and placed on the server, consists of an audio stream (typically a highly compressed, low quality sampled MP3 file, though it could be WAV or other popular audio formats), and a set of JPEG images which correspond to frames of the video at different points in time. These components are retrieved from the Web either before the video is run (in which case they will be cached for use at runtime), or during runtime, at which point they are cached for later use. Therefore, the server, upon processing the video stream into these subcomponents, must publish them to a web server, along with some meta data (such as the number of "compression levels" and start/end times for each frame at each level)

Adjustable Based on Bandwidth- The client adjust its downloading strategy based on the available bandwidth, to switch to different compression levels offered by the server. A compression level is defined as a set of key video frames, a subset of the overall sequential list of JPEG images from a broken-down MPEG video, where each member of this subset is declared to persist over a specific time range.

Cache – Videos, or portions thereof, that were previously downloaded should be stored locally for later use, in an effort to eliminate duplicate downloading. The cache should ideally store all levels of compression for a given video, and provide the best available compression level in response to any frame request. At the same time, the cache should abstract all methods of storage from the player, and simply provide the player with the location on disk of the JPEG frame file to play.

Cache Controller – The client intelligence that allows the users to stay synchronized.

3.3 Implementation Details

In order to get the system to work, we created a small UI within CHIME (see section 4 for figures) that activated a hook that we added into the 3D client. When activated, it would deploy a screen/portal on wall of the room that the client's avatar was in so as to display the downloaded frames within it. Each client was also gives a small cache where they could store pre-fetched video, several probes to monitor the various variables that would control synchronization as well as a cache controller.

The probes included a cache monitor, a bandwidth monitor and a monitor that stated the exact location of the video a client was watching. These are software probes [10] that gather simple metrics and send them back to the cache controller for evaluation, over the publish-subscribe event notification system. As pointed out before, each client sends position updates via a UDP stream to all fellow clients in neighboring rooms so that fellow clients could render avatars in their respective accurate positions. The CHIME servers as well as the Video server note all the clients that start up any given video and assume that they are part of the same student group that wishes to watch the video. Updates about time index of the video that a client is watching is sent to all the other clients in the group.

Before the video even starts, the client tries to ascertain whether the user wishes to watch a particular video by looking up the workgroup calendar and starts to pre-fetch the highest density of frames from the video server so as to provide the best possible video experience. The pre-fetching module is the same component activated when a client pauses a video allowing the client to buffer the next few frames in the idle time.

The cache controller gets information about the contents of the cache, i.e. about the availability of extra frames in the timeline, as well as the position in the video and the current bandwidth (calculated by a simple ping to the server). The cache controller, since having already parsed the hierarchy of frames available in every compression level (gotten by downloading a pre-determined structured document about the frames), makes a decision about which frame to download next in the available time between current time and the time when that frame will be displayed based on available download. The cache controller also knows the duration for which each from will be displayed on the client's screen and uses this information to try and optimize on the level and density of frames to be downloaded. Any pauses by the client are simply utilized to download the highest quality and density of frames possible before the client restarts the video again.

CHIME clients synchronize with one another (peer-to-peer) by sending a time index in the UDP stream at least once every 0.33 seconds. Therefore, our aim was to keep the client always synchronized within 0.33 seconds of one another. If any client got out of sync with the others, the cache controller for that client would either instruct the client to lower or raise the level of frames that were being downloaded.

All VCR functions like play/stop and pause events were sent on the event bus since they were more major events that required action rather than just adjusting. They were also events that needed guaranteed action, something that a UDP packet cannot guarantee. All the clients play, stop or pause depending on the event sent out.

A workflow engine [18] is typically centralized and our workflow engine here had to keep the client in synchrony. Since that was the cache controller's job, the cache controller served as the workflow engine for this project. We found that even though the cache controller was decentralized, it provided us with good results because the logic control for each cache controller was the same. Results of our tests are in Section 4.

4 Testing and Results

We used a test bed of up to 10 clients ranging from 400MHz laptops on a 56Kbit modem up to a 3GHz machine on a 100Mbit network. The resulting experiment kept the videos synchronized between all 10 clients within an error of approximately 4.38 seconds (for the first 7 minutes of the video), i.e. at no point was any client more than 4.38 seconds apart from any other. However, at this point, the system started showing more of a disparity especially on the laptops that do not have native 3D hardware support built in and therefore have to render the virtual environment in Software mode, thus slowing them down further. Figure 3 shows the extremely small variance between the various clients through the entire video while Figure 4 shows that even when we had a test bed of ten clients, they were essentially synchronized through the entire video content.

Some points to note during our test –

1. We started all the client's videos together. We did not attempt to have a client start significantly after the others to that it could "catch up" with the rest.
2. Our tests did not include any handheld devices. However, as long as a CHIME client would run on a handheld device and the PDA has internet connectivity, the synchronization should work in the same way.
3. We noticed that there was tremendous network congestion during the test. After investigation we found that the previously sparse traffic on account of the UDP streams had gone up tremendously. We found that since the position update events were relatively rare, when we used UDP streams for synchronization, the $O(n^2)$ streams (where n is the number of clients) with updates sent every 0.33 seconds from each client to every other client caused a substantial amount of traffic on the network.
4. We found the 3D client of CHIME to be an extremely heavy weight system that took up a lot of system resources on even the fastest machines used in our test.

Therefore each system found it hard to cope with simple task like parsing of synchronization data.

5. Related to the above point, we found that the system stopped working after 7 minutes of run time on account of running out of system resources.

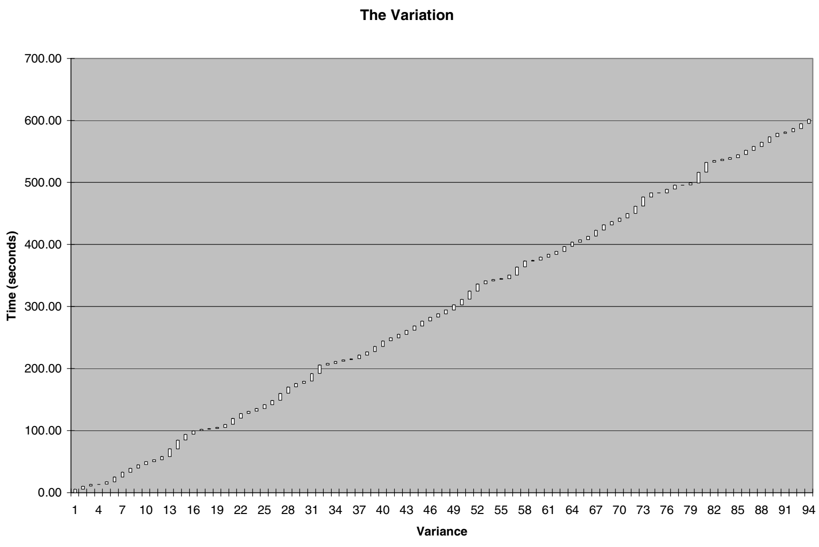

Fig. 3. Average variance of frames over time between clients

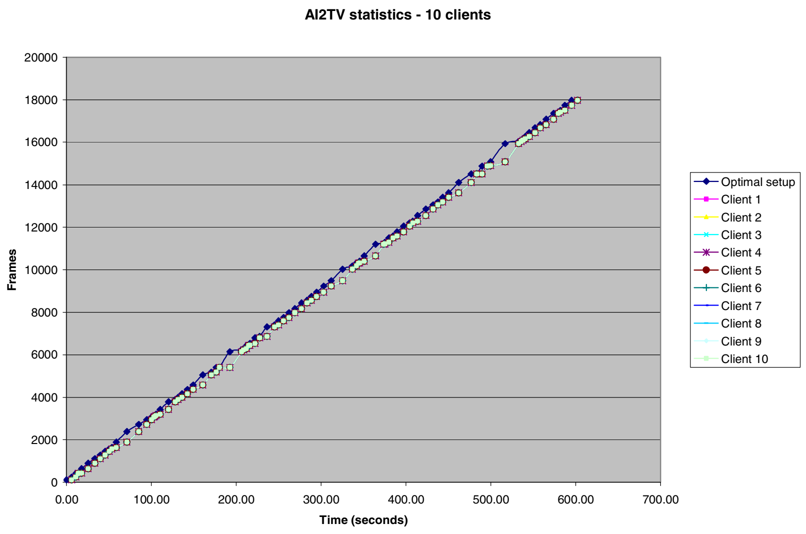

Fig. 4. Performance of 10 clients

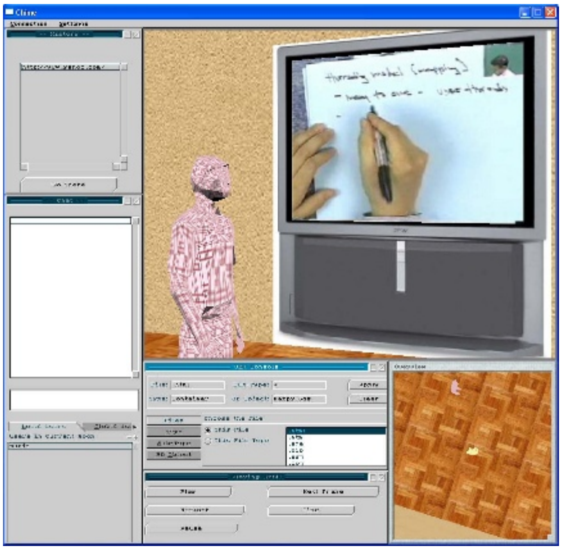

Fig. 5. VECTORS screenshot showing the video and team member

In Figure 3, we map the variance in frames vs. the amount of time taken to show them across the various clients and notice that the variance between two clients did not go over 4.5 seconds. Therefore, at no time were two clients more than 4.5 seconds apart.

Overall, our results show that with the video synchronization works as well as the collaborative tools available. VECTORS was extremely dependent on the stability of CHIME. However, stability issues aside, the system made for an excellent environment for enriching the educational experience. In small lab tests, simulated groups could collaborate on videos well and since VECTORS operated on a highly configurable pedagogical environment, the groups were able to access relevant educational materials when necessary (or prompted by the video). VECTORS successfully supported *synchronized* viewing of lecture videos, and allowed VCR functions like pause, rewind, etc. to operate in synchrony while discussing the lecture material via “chat”. VECTORS was successfully able to attend to the needs of the technologically disenfranchised, i.e. those with dialup or other relatively low-bandwidth networking.

5 Conclusion

We had presented a system, VECTORS, for the integration of lecture videos, with video synchronization, into a low-bandwidth virtual environment specifically designed for virtual classrooms for distance learning students.

This system has been designed as a plug-in to the previously developed collaborative virtual environment (CVE), CHIME, for small-group virtual classrooms.

VECTORS uses a peer-to-peer synchronization approach to support group viewing of lecture videos. By utilizing this approach, we have found that groups of co-located or non-co-located students can work together on group based assignments. In order to cater to group members with low bandwidths, instead of going with traditional approaches that involve skipping every n^{th} frame of a video, VECTORS employs semantically compressed and pre-canned videos and adjusts the clients among various compression levels so that they stay semantically synchronized. The videos are displayed as a sequence of JPEGs on the walls of a 3D virtual room, requiring fewer local multimedia resources than full motion MPEGs. As the results demonstrate, we have achieved a high degree of synchrony and have thus created a robust and useful pedagogical environment.

Acknowledgements

The Programming Systems Laboratory is funded in part by National Science Foundation grants CNS-0426623, CCR-0203876, EIA-0202063, EIA-0071954 and CCR-9970790, and in part by Microsoft Research.

References

[1] C. Bouras, A. Philopoulos, Th. Tsiatsos, “e-Learning through Distributed Virtual Environments”, J. of Network and Computer Applications, July 2001.

[2] Christos Bouras, Dimitrios Psaltoulis, C. Psaroudis, T. Tsiatsos “An Educational Community Using Collaborative Virtual Environments”. ICWL 2002: 180-191

[3] Dan Phung, G Valetto, Gail Kaiser, “Autonomic Control for Quality Collaborative Video Viewing”, Computer Science Dept., Columbia University TR# cucs-053-04

[4] J. Walpole, R. Koster, S. Cen, C. Cowan, D. Maier, D. McNamee, C. Pu, D. Steere, and L. Yu. A Player for Adaptive MPEG Video Streaming Over The Internet. In 26th Applied Imagery Pattern Recognition Workshop. SPIE, October 1997

[5] Stephen E. Dossic, Gail E. Kaiser, “CHIME: A Metadata-Based Distributed Software Development Environment”, Joint 7th ESEC Conference and 7th International Symposium on the Foundations of Software Engineering, Sept. 1999

[6] S. Dossick, “Groupspace Services for Information Management and Collaboration”, PhD Thesis, Columbia University, November 2000

[7] Stefan Fiedler, Michael Wallner, Michael Weber, “A Communication Architecture for Massive Multiplayer Games” Postion Paper, NetGames 2002

[8] Ekaterina Prasolova-Forland, “Supporting Social Awareness in Education in Collaborative Virtual Environments”, Int. Conf. on Engineering Education, 2002

[9] Ekaterina Prasolova-Forland, “Supporting Awareness in Education: Overview and Mechanisms”, In proceedings of ICEE, 2002

[10] Philip N. Gross, Suhit Gupta, Gail E. Kaiser, Gaurav S. Kc and Janak J. Parekh, “An Active Events Model for Systems Monitoring”, Working Conference on Complex and Dynamic Systems Architecture, December 2001

[11] Gail Kaiser, Giuseppe Valetto, “Ravages of Time: Synchronized Multimedia for Internet-Wide Process-Centered Software Engineering Environments”, 3rd ICSE Workshop on Software Engineering over the Internet, June 2000

[12] J. Liu, B. Li, and Y.-Q. Zhang, “Adaptive Video Multicast over the Internet”, IEEE Multimedia, Vol. 10, No. 1, pp. 22-31, January/February 2003

[13] T. Liu, J. Kender, “A Hidden Markov Model Approach to the Structure of Documentaries”, Content-Based Access of Image and Video Libraries, 2000

[14] http://i-lab.mit.edu

[15] S. Benford, D. Snowdon, C. Greenhalgh, “VR-VIBE: A Virtual Environment for Co-operative Information Retrieval”, Computer Graphics Forum, 1995

[16] Thanasis Daradoumis, Fatos Xhafa, Joan Manuel Marquès, “Evaluating Collaborative Learning Practices in a Virtual Groupware Environment”, CATE 2003

[17] K. Rothermel, T. Helbig, “An Adaptive Protocol for Synchronizing Media Streams”, Multimedia Systems, Volume 5, pages 324-336, 1997

[18] Jason Nieh, Monica S. Lam, “A SMART Scheduler for Multimedia Applications”, ACM Transactions on Computer Systems (TOCS), 21(2), May 2003

[19] J. Liu, B. Li, Y.Q. Zhang, “Adaptive video multicast over the internet” IEEE Multimedia, 10(1):22-33, January/March 2003

[20] H. Liu, M. E. Zarki, “A synchronization control scheme for real-time streaming multimedia applications”, In Packet Video, April 2003

[21] A. J. Gonzalez, H. Adbel-Wahab, “Lightweight stream synchronization framework for multimedia collaborative apps”, Comp. and Communications 2000

[22] D. Ferrari, “Design and application of a delay jitter control scheme for packet-switching internet works”, In 2nd International Conference on Network and Operating System Support for Digital Audio and Video, pages 72-83, 1991

[23] J. Escobar, C. Partridge, and D. Deutsch, “Flow synchronization protocol”, IEEE Transactions on Networking, 1994

[24] A. Campell, G. Coulson, F. Garcia, and D. Hutchison, “A continuous media transport and orchestration service”, In SIGCOMM92: Communications Architectures and Protocols, pages 99-110, 1992

[25] http://unreal.epicgames.com

[26] Suhit Gupta, Gail Kaiser, "A Virtual Environment for Collaborative Distance Learning With Video Synchronization", CATE, March 2004