Frame Rate Exclusive Sync Management of Live Video Streams in Collaborative Mobile Production Environment

Mudassar Ahmad Mughal
Mobile Life @ Stockholm University
Box 1197, SE-16426 Kista-Sweden
mamughal@dsv.su.se

Goranka Zoric
Mobile Life @ Stockholm University
Box 1197, SE-16426 Kista-Sweden
goga@mobilelifecentre.org

Oskar Juhlin
Mobile Life @ Stockholm University
Box 1197, SE-16426 Kista-Sweden
oskarj@dsv.su.se

ABSTRACT

We discuss synchronization problem in an emerging type of multimedia applications, called live mobile collaborative video production systems. The mobile character of the production system allows a director to be present at the site where he/she can see the event directly as well as through the mixer display. In such a situation production of a consistent broadcast is sensitive to delay and asynchrony of video streams in the mixer console. In this paper, we propose an algorithm for this situation called "frame rate exclusive sync manager", which draws on existing reactive source control synchronization techniques. It relies solely on frame-rate control and maintains synchronization between live video streams while ensuring minimal delay by dynamically adapting the frame-rate of the camera feeds based on synchronization offset and network bandwidth health. The algorithm is evaluated by simulation which indicates algorithm's capability of achieving increased synchronization among live streams.

Categories and Subject Descriptors
H.5.1 [**Information interfaces and presentation**]: Multimedia Information Systems –*video*

General Terms
Algorithms, Measurement, Performance, Experimentation

Keywords
Synchronization, production, video, mobility, live broadcast, streaming, collaborative, frame rate adjustment, network delay.

1. INTRODUCTION

We conduct design oriented research [31] in the area of mobile video services and applications. Traditionally, this type of research approach combines studies of user experiences in combination with an understanding of emergent technical research, to generate interesting and meaningful prototype applications. The research prototypes built are developed only to influence the understanding of the pros and cons of the applications suggested, and to further develop understanding of user experiences.

This method has been used to produce two different collaborative live video mixers [8, 9, 10, 11] which allow users to produce videos collaboratively using multiple mobile cameras, in a manner similar to how professional live TV production teams work, and stream the resultant video live for a (public) viewing. Such system consists of mobile cameras capable of live streaming via 3G/4G mobile networks, a mixer console and a webpage displaying a final video output. Mobile cameras stream live video of the event being filmed to the mixer console. The mixer console receives live video streams from all the mobile cameras and shows them simultaneously on the screen, enabling the director (a user controlling the mixer console) to "multi-view" all available content. The task is then to decide, on a moment-by-moment basis, which camera to select for the live broadcast. Based on the director's selection, the final video output is made available on the webpage for the consumption in real-time. For more details about such systems please see [11].

We learned that the availability of a mobile mixer device generates a new type of delay and synchronization problem i.e. the delay between what the director can see herself of an activity and when it appears on the mixer screen. The problem has not existed in previous "OB-bus" TV production system since the directors then mix while sitting in a production room and only look at the camera feeds in the mixer console. Now, light-weight mobility of this sort of equipment allows users to produce content using the mixing system while staying on site of the even that is being filmed and he/she can observe the event directly as well as through live camera feeds in the mixer console (referred to as "in-view mixing") [21]. This is where the new and interesting challenge emerges, as being witnessed in field studies with mobile vision mixers [11]. In this case the director can notice delay between camera feeds showing the event and the event per se, and thus high delays causes critical problems with production of a consistent live broadcast. We suggest that the emerging mobile systems must account for this technical challenge, and suggest a more detailed investigation. We suggest that the next step, which is presented here, is to articulate the details of this problem and make an addition to the design of such systems.

Since complete implementation is relatively time consuming we therefore suggest to take a middle step which includes a specification of the problem, as well a simulation of proposed solution for new type of synchronization problem in such applications. We would then get early indications of the possibility to handle this problem. There are two problems affecting the work of the director at the mixer console caused by end-to-end delay of video streams (time that is taken to transmit the packet through the network from a source to a destination):

1. In the mobile video production systems, the director often can choose between looking at the event per se and at the camera feeds of it ("in-view mixing"), when making broadcast selections. Due to delays of video streams, the time for the actual selection of a cut, as decided by looking at video streams in the mixer console, is not aligned with the event per se. This makes it difficult to fine-tune the switch from one view of a situation to another, such as when moving from an overview shot to a detailed shot, during a particularly interesting situation.
2. Due to the architecture of the Internet, delay from each camera is potentially going to be different and will result in asynchrony in the live feeds presented to the mixer. In the case when all the cameras are filming the same event from different angles, which is likely in collaborative production, the inter-camera asynchrony will affect director's multi-viewing, causing the same sort of problems in visual story telling as in the previous case.

If not handled correctly, both of them lead to problems for the director's storytelling, and consequently they will influence the perceived experience of the final broadcast. Thus, low delay and synchronization in between video feeds and the event are important requirements for *in-view* mixing in live mobile collaborative production systems. Our proposed solution called "frame rate exclusive sync manager (FESM)" keeps the delay as low as possible by avoiding buffering for synchronization, addressing the first problem; and maintains synchronization between video streams by dynamically adapting the frame rate of the camera feeds according to the bandwidth health, addressing the second problem.

Existing live collaborative mobile production systems like [3, 8] do not address synchronization and delay problems in the mixer console. In professional live TV production there is a delay of several seconds between the event and when it reaches the viewers in their homes. This divergence is almost never experienced as a problem. However, in the actual production situation, i.e. when the video systems are collaboratively tied together, the demands on low delays and synchronization are high. Delays in the professional live TV production environment are minimized by high speed dedicated transmission media and specialized hardware to synchronize multiple cameras. On the other hand, delay between the mixing moment and the event per se is of no consequence in professional TV systems, because of the physical separation between the event and the production environment.

Buffering techniques are commonly used for inter-stream synchronization with smooth presentation [7, 12, 24]. However, extensive buffering causes increased delays, making this method inapplicable for *in-view* mixing. In this paper we focus on the solution for synchronization of video streams in *in-view* mixing scenario where minimal delay is required.

Our proposed solution relies solely on the dynamic frame rate control at the video generation source for achieving synchronization between video streams. It does not rely on skipping frames and buffering at the receiver side for inter-stream synchronization, and thus avoiding addition of extra delay. We preferred frame rate reduction based solution in comparison with (a) spatial resolution reduction or (b) increasing the compression rate, because approaches (a) and (b) lower the visual image quality which is undesirable in the "in-view mixing" context. This work is an extension to the work done in [21] which discusses context based software solutions to handle the problem of synchronization and delays in live mobile collaborative video production systems. While [21] offers more general analysis and discussion about the new interesting problems and how they can be handled, this work focuses on a more concrete solution specifically for "*in-view* mixing". We propose, and evaluate by simulation an algorithm that maintains synchronization between continuous media streams with minimal delay by dynamically adapting the frame rate of the camera feeds based on the bandwidth health. The simulation results show that the algorithm successfully synchronizes multiple video streams keeping the synchronization offset under 140 milliseconds (ms) which is below the typical requirement according to [12]. Though, the reduced frame rate results in less smooth video presentation which can be tolerated in *in-view* mixing mode [21].

The rest of the paper is organized as follows. Background and related work are presented in section two. The description of the proposed solution, as well as of the algorithm is presented next. In section four, the results of the algorithm simulation are presented. Finally, we discuss the results and conclude the paper.

2. BACKGROUND AND RELATED WORK

Solutions for media synchronization in different scenarios have been proposed and they can be divided into three categories : Intra-stream synchronization, Inter-stream synchronization and Group synchronization[5].

In our case, the mixing is dependent on inter-stream synchronization, i.e. on methods that ensure synchronization among multiple live streams. Many studies have focused on synchronization of multiple live video streams. Works in [16, 17, 25, 27] have proposed solutions for inter-stream synchronization based on the VTR (virtual time rendering) algorithm that involves changing the buffering time according to the delay estimation. Bartoli et al. [2] suggest a synchronization scheme that ensures inter-stream and intra-stream synchronization for videoconferencing services over IP networks by preventive modification of the length of the silent periods for intra-stream synchronization. Similarly, authors in [7, 12, 23, 24] proposed different solutions that focus on inter-stream synchronization of live media streams using techniques like play out duration extension or reduction, reactive skips and pauses and frame duplication. As all of these solutions involve at least some level of buffering and thus introduce extra delay, such synchronization solutions do not suit our requirements for *in-view* mixing.

There are studies, e.g. [1, 15, 20, 29] which make use of reactive source control schemes where the video source or sender adjusts the rate of transmission in reaction to the detected asynchrony in order to achieve synchronization. For example Huang et al. [15] propose a scheme in which the source decreases its transmission rate when the network is congested and slowly increases its transmission rate when the network congestion is over. On other hand, if the source detects that recovery of synchrony is difficult for the receiver, it can decrease the number of media streams transmitted [18, 19, 22] Although, all above mentioned works make use of reactive source control techniques, none of them or any other (to the authors' best knowledge) provides a solution that relies solely on transmission rate control to cope with the varying bandwidth for achieving inter-stream synchronization. The existing approaches make use of the combination of the transmission control technique with frame skipping and duplication at receiver side, and other buffer control mechanisms which introduce additional delay. Considering the use of variable bitrate encoding (VBR) for which amount of output data per time segment varies, it takes more time to encode since the process is more complex and is not good for our delay sensitive application.

Summing up, there is a large body of research that addresses the topic of inter-stream synchronization in variety of situations. However, none of the related works addresses synchronization problems in between video streams in the services such as collaborative mobile live video production where minimal delay is required when the director is present at the site of event that is being filmed ("in view" mixing). We take our inspiration from the above mentioned works that use reactive source control techniques and propose a synchronization algorithm that meets the requirements of "in-view mixing" scenario.

3. PROPOSED SOLUTION

On site delay and asynchrony obstruct and confuse the director. Smoothness in video feed, however, may be compromised since the director can also directly see and observe the event. The requirement here is thus low delay, and high synchronization. To meet up such requirement of *in-view* mixing scenario, we propose frame rate exclusive synchronization manager.

Live Mobile Video Streaming When streaming live videos from mobile phones using 3G/4G connections, the available bandwidth is not guaranteed. We can experience fluctuations in the available bandwidth. Thus, if we are broadcasting the same event using two or more mobile phones, individual streams may experience different delays over the network, which, in turn, causes asynchrony among them. Here we want to note that latency in video coding which also could affect synchronization is not taken into consideration in this work, but we rather concentrate only on delay caused by variations in the available bandwidth. Let's suppose we are streaming live video from the same event with two mobile camera sources (S1 and S2, fig. 1) to corresponding receivers (R1 and R2) in a video mixer console. The vertical bars in the streams in fig. 1 represents frames that are captured at the same instant. Now suppose the network link from S1 is slower than from S2. This difference will cause video stream from S1 to be delayed, thus resulting in asynchrony when video is presented in the mixer console. The aim of our solution is to allow speeding up the video frames despite of lower bandwidth, so that both streams can be presented in the mixer console synchronized.

When we stream video from one point to another, we do it on a certain frame rate. The standard approach in streaming services is that a frame rate is negotiated in the beginning of a streaming session and remains the same during the rest of the session. Let's suppose the negotiated frame rate between video source and receiver is 15 fps. This means, 15 frames will be used to cover the event lasting one second. This requires the same amount of data to be transmitted over both the slow and the fast link to cover the same amount of time. When slower link does not support required amount of data in one second (bitrate), the frames are delayed. However, if we reduce frame rate to, e.g. 8 fps, the same amount of time (one second) will be covered with fewer frames. By requiring less data to be transmitted over the link over the same amount of time, the frames will appear as streaming faster. An obvious drawback of this approach is a loss of smoothness in the video playback. However, in "*in-view*" *mixing*, smoothness is not a priority requirement.

3.1 Frame rate exclusive sync manager

Figure 1 shows mobile cameras S1 and S2 sending live video streams to receivers R1 and R2 in the mixer console. The vertical bars in streams represent video frames that are captured at the same moment. Live video stream transmitted from S1 to R1, denoted as "stream *i*" is delayed as compared to "stream *j*", the live video stream transmitted from S2 to R2. The "Frame rate exclusive Sync Manager"(FESM) is a component in the mixer console that reads time stamp information available in video streams and determines which stream is out of synchronization, i.e. delayed, by calculating the synchronization offset. If the offset is too big, the FESM signals the corresponding video source to drop the frame rate. FESM also keeps track of the network bandwidth and in the case the bandwidth is recovering, it signals the corresponding receiver to recover the frame rate. There are existing techniques available in literature like [6, 26, 30] that can be used for monitoring the bandwidth. In this work, it is assumed that the bandwidth monitoring function is already in place.

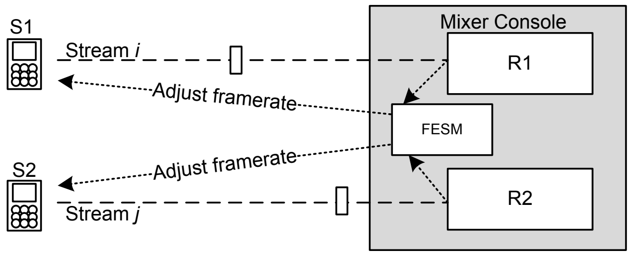


Figure 1: Frame rate exclusive sync manager

3.1.1 Clock Synchronization

It is assumed that clocks in the mobile cameras and receivers are synchronized using the network time protocol (NTP), and that each video frame in video stream is time stamped. NTP is capable of synchronizing the clocks with the accuracy of the range of few milliseconds [12]. To keep all streams synchronized, we compare time stamps in individual streams to a single reference clock, and try to keep each stream synchronized with the reference clock. *The reference clock* is the clock in the receiving system (the mixer console in this case), which is synchronized as well with all the senders. The reference clock generates time stamps T_c with frequency equal to maximum supported frame rate, e.g. 25 frames per second. Synchronization of a video stream with the reference clock goes as follows.

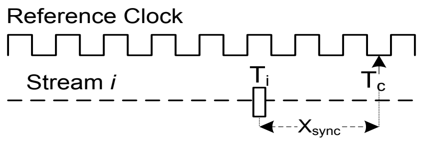


Figure 2: Calculation of Xsync in FESM

Let's say T_i is the timestamp on the frame received in R1 from S1. When video frames arrive at their corresponding receivers, the FESM (see Figure 1) reads T_i and calculates the synchronization offset $Xsync_i$ by computing the difference between the current value of the reference clock T_c and the time stamp on received frame T_i (see Figure 2). Thus, $Xsync_i$, calculated as

$$Xsync_i = |T_c - T_i|$$

Where $Xsync_i$ is the synchronization offset of stream *i*. In multimedia systems synchronization requirements among streams can range from at least as low as 100ms to approximately 300ms [12].

Algorithm Figure 3 shows the flowchart of the continuous loop for the proposed algorithm. After the algorithm is started, parameters *Thresh* and *Ref Clock* are initialized. *Thresh* is a synchronization threshold for the synchronization offset *Xsync*, and *Ref Clock* is the reference clock. The algorithm proceeds after reading the time stamp of the received frame T_i and the link bandwidth *B/W*. If *B/W* is recovering, the control will recover the frame rate to normal and jump back to the point in the algorithm after initialization, and if *B/W* is getting worse or not recovering, then synchronization offset Xsync is calculated (time stamp T_c is obtained from the *Ref Clock*). Next the *Xsync* is compared to the synchronization threshold *Thresh*. If *Xsync* is larger than *Thresh*, the frame rate is dropped by a given step value at the sender, and iteration starts over, otherwise the iteration is immediately started over.

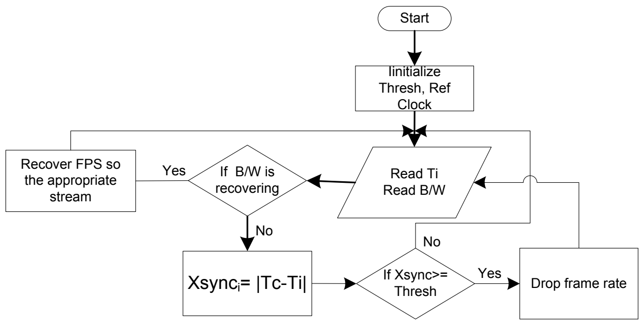


Figure 3: Flowchart of the proposed algorithm

In the case of multiple streams, every stream is independently handled and its frame rate is adjusted dynamically to keep it synchronized with the reference clock. When all the streams synchronize to one reference clock, they are automatically synchronized.

Evaluation by simulation We simulated its performance to get early feedback on how well it maintain synchronization between video streams in a case where the bandwidth changes over time, when other parameters like encoding delay are constant. The synchronization offset and the frame rate of video streams were measured and plotted.

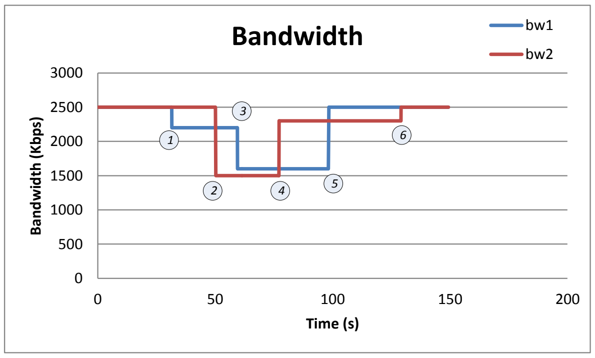
We simulated two video sources, two receivers, the network link and the frame rate exclusive synchronization manager (FESM). We generated time stamped video frames from two video sources at the same time and transmitted them as two separate video streams to separate receivers. We performed the experiment with two video streams, during which we were changing available bandwidth of the emulated network links, and observed how it influenced synchronization offset and stream frame rate.

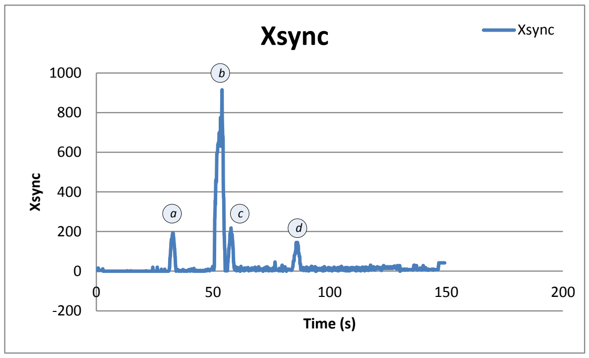
We implemented video senders and receivers in Max/MSP/Jitter [13]. Video is encoded using SPIHT (Set Partitioning in Hierarchical Trees), and the streams are sent to the receivers via TCP. We used a short video with 176x144 spatial resolution which was recoded at 25 frames per second. The network link between senders and receivers were emulated using ipfw [14]. The ipfw is a utility in the Mac OSX that works as a user interface for controlling a dummynet traffic shaper which allows us to throttle available bandwidth for the specific port number and IP address [28]. The FESM was also implemented using Max/MSP/Jitter. When the frame rate is dropped by one step, it does not affect the Xsync value immediately because of network delay. Thus, in our simulation, the algorithm iterates every three frame using a mean value of last three Xsync values. In the simulation, we used 140 ms as the synchronization threshold (*Thresh)*, and *two* was used as the step value for the frame rate drop.

3.2 Results

Figure 4 show results of the experiments. Part (a) shows available bandwidth of the emulated network links was changing with respect to time. Part (b) presents synchronization offset (Xsync) changes in time, and parts (c) show frame rate(s) over time.

Experiment: Two video senders stream video via two separate emulated network links whose bandwidth can be controlled. We refer to individual streams as "stream 1" and "stream 2" and their respective network links are referred to as "link 1" and "link 2" in this text from now on. We ran the simulation for 150 seconds. The available bandwidth for both streams is initially set to 2500 kbits/s as shown in the Figure 4a. Both streams are initially running at the frame rate of 25 frames per second (see Figure 4c). After almost 30 seconds, the available bandwidth on link 1 falls to 2200 kbits/s (see point *1* Figure 4a). Consequently the stream 1 is delayed and Xsync value rises to 224ms (see point *a* in Figure 4b). As we are interested to see how two streams synchronize, the Xsync shown in Figure 4b is actually synchronization offset between stream 1 and 2. We know that





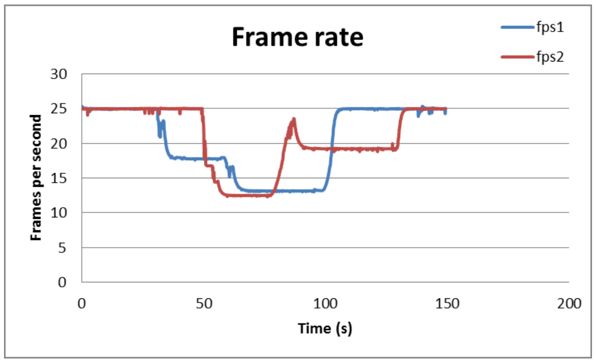


Figure 4a-c: Results of the second experiment

$$Xsync_i = |T_i - T_c|,$$

where $Xsync_i$ is the synchronization offset of stream 1 with respect to the reference clock. Also,

$$Xsync_j = |T_j - T_c|,$$

where $Xsync_j$ is the synchronization offset of stream 2 with respect to the reference clock. Hence, the synchronization offset between streams is:

$$Xsync = Xsync_i - Xsync_j$$

$$Xsync = |T_i - T_j|.$$

As soon as Xsync exceeds the *Thresh* value, the frame rate on stream 1 is dropped gradually (to 18 fps) until Xsync falls back within the threshold limit (140ms). The small fluctuations in the frame rate are caused by the variations in processing load of the simulation equipment. Later on during the simulation, the bandwidth available for stream 2 also falls to 1500kbits/s (see point 2 in Figure 4a). The Xsync again rises as high as 937ms as indicated at point *b* in Figure 4b. The algorithm handles this situation again by dropping the frame rate of stream 2 in the similar manner as we described for stream 1. Now both streams are running at the lower frame rates until the point *3* (Figure 4a) when the bandwidth of link 1 further drops to 1600 Kbits/s causing a gradual rise in Xsync to 248 ms (point *c* Figure 4b). The algorithm handles synchronization again by lowering the frame rate of stream 1 further to 13 fps and thus Xsync falls back within the threshold limit again within 1.5 seconds. After few seconds bandwidth of stream 2 (which was 1500 Kbits/sec until then) recovers to 2300 Kbits/s (see point *4* Figure 4a) and correspondingly the frame rate on stream 2 is also rises gradually to 22 fps which causes Xsync to exceed the threshold (see point *d* Figure 4b). Consequently the algorithm settles frame rate to 19 fps where Xsync is under threshold value. Later, at the point *5,* the bandwidth of the link 1 recovers to 2500 Kbits/sec which causes the recovery of the frame rate in stream 1 to 25 fps. Frame rate of stream 2 is also recovered to 25 fps when its bandwidth rises to 2500 Kbits/s later in the simulation (point *6* Figure 4a).

4. DISCUSSION

Here we discuss how well FESM handles synchronization in general as well as in specific balancing of parameters such as synchronization recovery time, step value, algorithm iteration size and their effect on algorithm.We also discuss implications of synchronization recovery time and resultant frame rate on quality of experience for a director using the mixer console.

Capability of FESM: The simulation of FESM as presented in the previous section showed that our algorithm is capable of achieving increased synchronization of multiple video streams with low delay despite varying bandwidth.

Synchronization recovery time: It is important to understand the length of the synchronization recovery time to evaluate the quality of the algorithm, and how does it influence the work of a director mixing in between video streams in an *in-view mixing* scenario. The synchronization recovery time is the time between the point when synchronization offset becomes larger than the threshold value and the point when it is again below the threshold. In our experiments, the average recovery time was 3.5 seconds, with synchronization offset ranging between 163ms and 937ms.

Step value: Synchronization recovery time in our experiments depends on the step value used for frame rate drop, and on how severe is the change in the bandwidth. If the higher step value was used to adjust the frame rate, synchronization was recovered in shorter time than if a lower step value was used. However, the use of the higher step value might result in a resultant frame rate lower than that with the lower step. On the other hand, too small step value results in long recovery time (e.g. having 1 as the value for frame rate drop resulted in 24 seconds in similar situation as point b in 4**Error! Reference source not found.**b). After trying with different step values for frame rate drop, we found that step value *two* as a good enough tradeoff.

Bandwidth fluctuations: Considering the bandwidth changes, if the fluctuations in between different video links are bigger, and thus resulting in the high synchronization offset, the more steps it will take to recover synchronization.

Algorithm iteration size: The fact that algorithm iterates every three frames also contributes to higher recovery time. We chose this iteration update to allow enough time for the system to reflect the synchronizing effect of the frame rate adjustment.

Experience of quality: It is interesting to discuss synchronization recovery time from the perspective of a director doing mixing. From the results presented in the earlier section we can say that FESM ensures increased synchronization without introducing additional delays in the streams at mixer console is a step further in improving the quality of experience for director in such systems being used in “in-view mixing” settings. However, it is visible that potentially high synchronization recovery time reveals challenges that we may have to deal with in real life implementation. When considering the *in view mixing* situation in a collaborative live mobile video production, where the director is producing a live broadcast, the bandwidth drop of one of the live streams and long recovery time may cause a problem for a director. The director may notice asynchrony in his mixer console. During that period there may be several important events happening, and the lack of synchrony may lead to wrong mixing decisions. This becomes even more sensitive in a case when viewers’ feedback is taken into consideration – in that case it is not only delays in between video streams in the mixer console and the event per se that matter, but also delay occurring in between the event, live broadcast of it, and the feedback. Also, it would be interesting to see how low the frame rate could be lowered not to influence the director’s work, i.e. what is the minimum image quality that these systems should enable. For this, a field study using a prototype with real users should be conducted.

Although, our simulation experiments prove the concept that increased synchronization can be achieved using FESM, we need to understand how big synchronization recovery time can be tolerated in practice in order not to influence the director’s decisions.

5. CONCLUSION AND FUTURE WORK

We proposed a synchronization algorithm for an *in-view mixing* scenario in live mobile collaborative video production applications i.e. a situation where the director can observe the event of filming in situ as well as through the live camera feeds in the mixer console. The proposed solution increases synchronization by dynamically adapting the frame rate at the video sources with bandwidth fluctuations. This method avoids buffering, and thus provides synchronization with minimal delay. The down side is that the video playback loses smoothness in mixer console when the frame rate is dropped to handle synchronization. As we focus on a specific scenario in mobile collaborative live video mixing systems where the director is present at the filming location, this drawback does not affect the director’s work. We evaluated the proposed algorithm by doing simulation tests and presented our results. The results showed that the algorithm handles synchronization with average recovery time of 3.5 seconds. Although this simulation study proves the concept and unpacks the influence of different parameters involved on synchronization, the implementation is needed to demonstrate the performance of the proposed solution in the real network with un-deterministic behavior, as well as to understand how long synchronization recovery time could be tolerated in order not to influence director's decisions in *in-view* mixing scenario. Therefore, the next step is a development of a prototype and a user study.

6. REFERENCES

[1] Ali, Z. Ghafoor, et al. Media synchronization in multimedia web using a neuro-fuzzy framework, IEEEJ. Sel. Areas Commun. 18(2)(2000)168–183.

[2] Bartoli, I. et al. A synchronization control scheme for videoconferencing services, J. Multimedia 1 (4) (2007) 1–9.

[3] Bentley, F., Groble, M. 2009. TuVista: meeting the multimedia needs of mobile sports fans. In Proc. Of MM '09.

[4] Blum, C. Practical Method for the Synchronization of Live Continuous Media Streams, Institut Eurécom.

[5] Boronat, F. et al., (2009) Multimedia group and inter-stream synchronization techniques: A comparative study, Information Systems 34 pages 108–131

[6] Breitbart Y. et al, Efficiently monitoring bandwidth and latency in IP networks, in: Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2001), Page(s): 933- 942 vol.2

[7] Correia, M. Pinto, P. Low-level multimedia synchronization algorithms on broadband networks, in: Proceedings of the Third ACM international conference on Multimedia, San Francisco, CA, USA, November1995, pp. 423–434.

[8] Engström, A. Juhlin, O. et al (2009) Instant broadcasting system: mobile collaborative live video mixing. In proc. SIGGRAPH ASIA '09 ACM Emerging Technologies

[9] Engström, A. Juhlin, O. and Reponem, E. (2010) Mobile broadcasting – The whats and hows of live video as a social medium, In Proc of Mobile HCI 2010, September 7–10, Lisbon, Portugal

[10] Engström, A. Perry, M. Juhlin, O. (2012) Amateur Vision and Recreational Orientation: creating live video together. In proc. CSCW 2012 Seattle.

[11] Engström, A. Zoric, G. Juhlin, O. et al The Mobile Vision Mixer: A mobile network based live video broadcasting system in your mobile phone, In proc. MUM 2012, Ulm

[12] Escobar, J. Partridge, C. Deutsch, D. Flow synchronization protocol, IEEE/ACM Trans. Networking 2 (2) (1994) 111–121.

[13] http://cycling74.com/products/max/ Accessed (13 Sept 2012)

[14] http://developer.apple.com/library/mac/#documentation/Darwin/Reference/Manpages/man8/ipfw.8.html Accessed (13 Sept 2012)

[15] Huang, C. M. Kung, H.Y. Yang, J. L. Synchronization and flow adaptation schemes for reliable multiple-stream transmission in multimedia presentations, J. Syst. Software56 (2) (2001) 133–151.

[16] Ishibashi, Y. Kanbara T. et. al., Media synchronization between voice and movement of avatars in networked virtual environments, in:Proceedings of the 2004 ACMSIGCHI International Conference on Advances in Computer Entertainment Technology, Singapore, June2004, pp.134–139

[17] Ishibashi, Y. et al. Inter-stream synchronization between haptic media and voice in collaborative virtual environments, in: Proceedings of the 12th annual ACM international conference on Multimedia, New York, USA, October2004, pp. 604–611.

[18] Ishibashi, Y. et al. Media synchronization and causality control for distributed multimedia applications, IEICE Trans. Commun. E84-B (3) (2001) 667–677.

[19] Little, T.D.C. A framework for synchronous delivery of time-dependent multimedia data, Multimedia Syst. 1 (2) (1993) 87–94.

[20] Manvi, S.et al. An agent based synchronization scheme for multimedia applications, J. Syst. Software (JSS) 79 (5) (2006) 701–713.

[21] Mughal, M. A. Juhlin, O. "Context dependent software solutions to handle video synchronization and delay in collaborative live mobile video production", In Journal of Personal Ubiquitous Computing (2013).

[22] Ravindran, K. Bansal, V. Delay compensation protocols for synchronization of multimedia data streams, IEEE Trans. Knowl. Data Eng. 5 (4) (1993) 574–589.

[23] Rothermel, K. Helbig, T. An adaptive protocol for synchronizing media streams, ACM/Springer Multimedia Syst. 5 (5) (1997) 324–336.

[24] Rothermel, K. Helbig, T. An adaptive stream synchronization protocol, in: Proceedings of the Fifth International Workshop on Network and Operating System Support for Digital Audio and Video, Durham, New Hampshire, USA, April 1995, pp. 189–202.

[25] S.Tasaka, Y.Ishibashi, Media synchronizationin heterogeneous networks: stored media case, IEICE Trans. Commun. E81-B(8) (1998)1624–1636.

[26] Selin P. Selin et al, Available Bandwidth Measurement Technique Using Impulsive Packet Probing for Monitoring End-to-End Service Quality on the Internet, in: 17th Asia-Pacific Conference on Communications 2011, pp. 518-523.

[27] Tasaka, S. Ishibashi, Y. A Performance Comparison of Single-Stream and Multistream Approaches to Live Media Synchronization E81-B (11)(1998)1988–1997.

[28] The dummynet project, http://info.iet.unipi.it/~luigi/dummynet/ Accessed on 19 Sept 2012.

[29] Zhang, A. Song, Y. Mielke, M. Mielke, NetMedia: streaming multimedia presentations in distributed environments, IEEE Multimedia 9 (1) (2002) 56–73.

[30] Zhu, H. et al, Predictable Runtime Monitoring, in proceedings of ECRTS '09. 21st Euromicro Conference on Real-Time Systems, 1-3 July 2009, pp. 173-183

[31] Zimmerman, J., Forlizzi, J., and Evenson, S.. Research through design as a method for interaction design research in HCI. In Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI'2007)