Download and Cache Management for HTML5 Hypervideo Players

Britta Meixner[1,2], Christoph Einsiedler[1]
[1]University of Passau, Innstrasse 43, 94032 Passau, Germany
[2]FX Palo Alto Laboratory, 3174 Porter Drive, Palo Alto, CA 94304, USA
meixner@fxpal.com, einsied@fim.uni-passau.de

ABSTRACT

Web videos are becoming more and more popular. Current web technologies make it simpler than ever to both stream videos and create complex constructs of interlinked videos with additional information (video, audio, images, and text); so called hypervideos. When viewers interact with hypervideos by clicking on links, new content has to be loaded. This may lead to excessive waiting times, interrupting the presentation – especially when videos are loaded into the hypervideo player. In this work, we propose hypervideo prefetching strategies, which can be implemented in players to minimize waiting times. We examine the possibilities offered by the HTML5 <video> tag as well as the Media Source Extensions (MSE). Both HTML5 and MSE allow element pre-fetching (video and additional information) up to a certain granularity. Depending on the strategy and technology used, beginning scene waiting times and the overall download volume may increase. The strategies presented in this paper allow the number of delays during playback and the overall waiting time of a video to be reduced significantly from an average of 8.1 breaks to less than one break. The overall waiting times can be reduced by one third, to less than 18 seconds improving the hypervideo watching experience.

Keywords

Pre-fetching; Download; Cache; HTML5; Media Source Extensions; Hypervideo; Quality of Experience

1. INTRODUCTION

Videos on the Web are becoming more and more popular. According to the Cisco Visual Networking Index, "mobile video traffic exceeded 50 percent of total mobile data traffic by the end of 2012 and grew to 55 percent by the end of 2014" [7]. "Globally, consumer Internet video traffic will be 80% of consumer Internet traffic in 2019, up from 64% in 2014" [8]. These statistics show how important video is on the Internet in daily life.

HT '16, July 10-13, 2016, Halifax, NS, Canada

DOI: http://dx.doi.org/10.1145/2914586.2914587

Technologies like HTML5, JavaScript, and CSS allow the creation of annotated, interactive, non-linear hypervideos. These consist of linked video scenes and may contain additional information like video, audio, images, text, and other media, which can be selected by the viewer via hyperlinks, or are displayed in parallel with the main video scenes. This type of video allows a high level of interaction and non-linear navigation during playback, where every user can chose her/his own path through the hypervideo. A detailed description of annotated interactive non-linear video can be found in [16, 17]. These hypervideos can be used in several scenarios, like instructional videos [29], medical training [21], and walks through cities or buildings [19]. In the remainder of this work, we use the term "annotated interactive non-linear video" as synonymous with "hypervideo".

Hypervideos have metadata which define the underlying scene graph and link structures between videos and additional information. The metadata are needed by the player before the playback of the video may start. Analyzing the structures described in these metadata, the next possible displayable contents are known. This knowledge can be used to pre-fetch and cache future elements (video and additional information) and make them available before the viewer selects them. Accordingly, the waiting times are reduced which results in a better user experience. Furthermore, breaks during playback can be limited or avoided. While it may be a suitable strategy to download and cache the whole hypervideo on a desktop computer with an unlimited internet connection, data plans for mobile phones often limit the amount of high speed data. Consequently, downloading a whole video where huge parts will not be watched is not an option. New strategies need to be found to provide a good viewer experience while keeping the overall download volume as small as possible. Experiments in a simulation environment showed promising results for several algorithms and strategies for download and cache management [16, 17], but no implementation for real world players exists. In this work, we adapt, rewrite, and implement the strategies and algorithms described in [16, 17] with HTML5 [36] and the Media Source Extensions (MSE) [37, 38]. Thereby, we make the following research contributions:

- We transform findings from a simulation framework for download and cache management in hypervideos into efficient pre-fetching strategies for real-world hypervideo web players using HTML5 and MSE.
- We outline the limitations that appear when using HTML5 and MSE for implementing hypervideo players for the Web.

- We show that using the right combination of framework and algorithm, we can achieve smaller waiting times before and during playback, as well as overall waiting times compared to implementations using the standard HTML5 video-tag. We show that we can limit the number of breaks during playback to zero in most of the times. While limiting waiting times and pauses we also achieve the smallest possible overall download volumes.
- We provide guidelines for future implementations in this area using HTML5 and MSE.

The remainder of this work is structured as follows: In the following section, an overview of related work in adjoining areas is given and discussed. Section 3 describes algorithms used and their implementation in different web frameworks. The results of a comprehensive evaluation with patterns most frequently appearing in hypervideos are described in Section 4. A summary of the results and an outlook on future work can be found in Section 6.

2. RELATED WORK

Related work exists in the areas of web video frameworks, hypervideo players on the Web, and pre-fetching strategies for hypervideos. To the best of the authors' knowledge no related work exists combining these three aspects. Therefore, we will discuss the most important work from all areas.

2.1 Web Video Frameworks

Frameworks and standards for embedding video into a website are SMIL [35], the Adobe Flash Player [3], Microsoft Silverlight [22], and HTML5 [36].

The first public release of the SMIL Specification was in 1997 [32]. Its final version (SMIL 3.0) from 2008 [35] "allows authors to write interactive multimedia presentations. Using SMIL, an author may describe the temporal behaviour of a multimedia presentation, associate hyperlinks with media objects and describe the layout of the presentation on a screen" [35]. "In June of 2000, Microsoft Internet Explorer version 5.5 was the first product to support SMIL 2.0 technology" [31]. A true subset of SMIL 2.0, SMIL Animation [33] in SVG is supported by Firefox 41, Chrome 46, Safari 9, and Opera 32 [1]. However, none of the current browsers supports the whole standard in its final version.

The first version of the browser plug-in Flash Player was released by Macromedia in 1997. The first version capable of playing video was Flash 6 released in March 2002 [39]. Buffering and caching of videos at player side is possible in current versions using ActionScript 3 [2].

Another browser plug-in is Microsoft Silverlight, which was released in September 2007 [23]. As with the Adobe Flash Player, Silverlight is also capable of controlling its cache up to a certain degree. A comparably new way of embedding video into a website is provided by HTML5 [36].

In contrast to the already described plug-ins, HTML5 is implemented by the browsers directly. While the video element was in the standard since the first working draft from January 2008 [34] it was not implemented in the browsers at that time. The first browsers capable of interpreting the video element were Safari 3.1 (2008), Chrome 3.0 (2009), Firefox (Gecko) 3.5 (1.9.1) (2009), Opera 10.5 (2009), and Internet Explorer 9.0 (2011) [24]. Predefined settings and states for pre-loading and buffering are available for the video element, but a fine grained manual control of the download and cache is not possible.

While Flash was widely used for the playback of video on the Web before the introduction of HTML5 video [4], the "general trend for Flash usage is downward" [41]. Throughout the Web, different prognoses can be found claiming that HTML5 will be the main method to embed video into a website [28, 30] in the future. Silverlight has never played an important role in video playback on the Web compared to Flash. Therefore, we will focus on an implementation in HTML5 in the remainder of this paper.

2.2 Hypervideo Player for the Web

With growing bandwidths in the Internet and the availability of HTML5 in all important browsers, more and more hypervideo players are available. Each of them provides different functions offering more or less liberties and possibilities to the authors of hypervideos. Since January 2010, YouTube uses HTML5 video [27]. The YouTube player [42] provides functions to link videos with each other using buttons. The video selected by the button then opens in a new web-page which ruins the perception of a self-contained presentation. Besides the linking of videos, text boxes and speech bubbles can be added to provide additional information. This player does not cache or pre-fetch contents due to its loading behavior.

Linius [15] is a Wordpress-theme [40] for linear video with additional navigational features. Extended navigation in the video is possible. Scenes can be skipped using a table of contents. Each "chapter" of a Linius video may contain videos, text, images, audio files, interactive graphics, and more. Linius executes an initial pre-fetch before playback. Thereby, all images of the presentation are pre-fetched as well as the video of the first chapter (if it exists). Videos and audio files of the remainder of the presentation are loaded when they are selected during playback. While the initial pre-fetch may take some time, its loading progress is displayed.

A very similar player to Linius is Pageflow [9]. It is a content management system designed for the creation of multimedia stories. The navigation in Pageflow is similar to Linius. Chapters can be skipped by using the mouse wheel, the arrow keys, or by using the table of contents. As in Linius, chapters may contain videos, text, image, audio files, interactive graphics, and more, but a temporal synchronization of the contents with a video is not possible. Pre-fetching works the same way as described for Linius, where only metadata are pre-fetched for future chapters.

The Klynt player [12] is used for the playback of videos created with the Klynt authoring tool. In contrast to linked videos on YouTube, the hypervideo is presented in one player without loading a new page. Navigation between videos and additional information is possible. Compared to the YouTube player, this player offers a huge variety of additional information like text, Wikipedia-articles, info-graphics, or maps. To the best of the author's knowledge, it is not possible to customize the caching behavior of this player.

The SIVA HTML5 Player (for a detailed description see [20], a revised version with a GUI redesign is described in [21]) is used for the playback of hypervideos exported from the associated authoring tool (SIVA Producer, described in [19]). This player provides different ways to navigate in the hypervideo like button panels (to select the next scene from the underlying scene graph), a table of contents, or a key-

word search. Additional information is synchronized and displayed with the video in panels in the player or as overlays over the video. No pre-fetching is implemented in the current version of the player, but the availability of all elements is checked before the playback of each scene starts.

Summing up, many players provide interactivity and non-linear navigation in (hyper)videos, but none of them provides a comprehensive download and cache management strategy covering the whole video.

2.3 Pre-fetching Strategies for Hypervideos

Pre-fetching strategies can mainly be found for the streaming of linear videos in multicast environments or on different types of underlying networks. Fei et al. [10] describe an active buffer management for partitioned video multicast systems. Their client pre-fetches segments from broadcast channels and is thus able to provide VCR actions to the viewers. Laraspata et al. [13] describe an algorithm for variable bit rate video transmission over UMTS networks, which helps to reduce delays by taking user interactivity into account. Both works only deal with interactivity in a linear video. None of them takes additional information that has to be displayed with the video into account.

Grigoras et al. propose an adaptive approach to "stream hypervideo that takes into account user behavior" [11] reducing network induced latency. Their "technique is based on a model provided by a Markov Decision Process approach" [11]. Doing this, they describe two different methods. These are tested under stochastic network conditions. Their approach only takes video into account. Additional information like images, text, audio files, or other videos which are played parallel to the main video are not taken into consideration.

A simulation framework for download and cache management algorithms in annotated interactive non-linear videos is presented in [16, 17]. Strategies for pre-fetching, scheduling, and downloading elements of annotated interactive non-linear videos are presented as well as algorithms for the deletion of elements from the cache. This frameworks allows the simulation of videos linked in a scene graph which are extended with additional information displayed with main video scenes. Different bandwidth and cache settings can be used for the simulations. An initial set of algorithms is tested using patterns that often appear in hypervideos, like sequences, cycles, and tree-like structures with branches. Some algorithms show promising results in the framework and will therefore be integrated in the SIVA HTML5 hypervideo player [20, 21] as described hereafter.

3. HTML5 HYPERVIDEO

In this section we will give a deeper overview of the underlying software and algorithms for this work. We first describe the existing HTML5 hypervideo player. Then we introduce the implementation of the algorithms as tested in our simulation framework [16, 17]. We then describe how we adapted, rewrote and implemented these strategies with recent web technologies (HTML5 and MSE) to integrate them into the existing HTML5 hypervideo player [20, 21].

3.1 HTML5 Hypervideo Player

The basis of this work is the SIVA HTML5 hypervideo player from our previous work [20, 21]. This player has a video area (Figure 1, left side) where the main video scenes

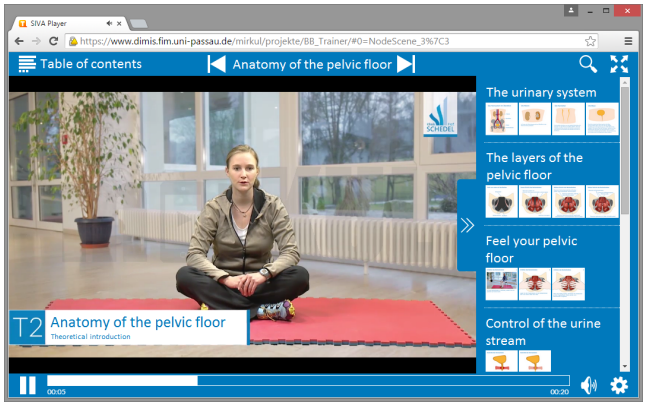

Figure 1: Screenshot of the SIVA HTML5 player with main video area and a fold-out panel for annotations.

are displayed. The annotation area 1, right side) may contain different types of annotations like images, image galleries, text, audio files, or additional videos. These annotations can be displayed for the whole duration of a scene or only for a defined time range within the scene. The annotation area in Figure 1 shows four image galleries. At the end of a scene, either the next scene is loaded automatically by the player (because the scene has only one successor) or a panel folds out from the right side which provides buttons for the selection of the next scene (in case the scene has more than one successor). The player provides all standard control elements as known from traditional video players at the bottom and additional control elements needed for hypervideos at the top of the player. These are a table of contents, buttons to jump to the next or previous scene, a keyword search, and a button for the full-screen mode. The structure of the video is read from a JSON file which is structured as described in [18].

This player provides all functions needed to read and process the metadata and display the media at the right place and time in the player. However, there are no pre-fetching functions implemented in this player so far. The buffering of each scene and its annotations starts when the scene is selected by the viewer at a button panel or when the scene is selected by the player in case of a sequence of scenes. How and when to buffer the media is left to the browser. This may lead to delays and scene transition waiting times.

3.2 Pre-fetching - Theory

As described in the previous section, each scene of a hypervideo consists of a main video (then again consisting of frames), and additional information which are displayed during a certain continuous range of frames in a scene. In order to provide all necessary contents at the time of display without further loading times (thus causing breaks in the flow during playback), strategies for pre-fetching and download scheduling are needed. We first implemented these strategies in a simulation framework to get a better feeling for the inter-dependencies between cache size, bandwidth, user interaction, and settings in a hypervideo like probabilities for choosing a certain scene, durations of scenes, numbers and sizes of annotations, sizes of the videos, and certain perennial structures each hypervideo contains. For an overview

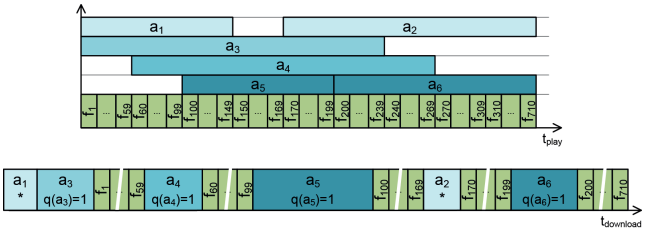

Figure 2: Playback schedule (top) and resulting linear download schedule (bottom) of a scene.

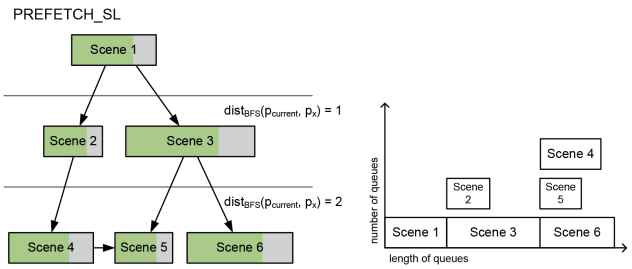

Figure 3: Download queues for the PREFETCH_SL strategy of a sub-part of a scene graph (left) with six scenes resulting in three layers (right).

of the framework and its results please refer to [16, 17]. We want to outline the most important concepts and algorithms from this work hereafter to show the difficulties and differences in a real-world implementation thereafter.

3.2.1 Scheduling a Scene

Taking a closer look at a scene, each one has two schedules: one for playback and one for download. The playback schedule of a scene containing six annotations and 710 frames is illustrated in Figure 2 (top). Annotations a_1 and a_3 are displayed from the beginning, then annotation a_4 is shown from frame f_{60} and so on. Taking a look at a possible linear download schedule, the annotations are scheduled before the frames they are displayed with to make sure they can be displayed immediately. This results in a schedule where annotations a_1 and a_3 are scheduled before frame f_1. Then all frames up to frame f_{59} are scheduled followed by annotation a_4 and frame f_{60} and so on. The resulting linear download schedule is shown in Figure 2 (bottom).

3.2.2 Scheduling a Sub-graph

With the graph structure of the scenes known from the metadata of a hypervideo, it is possible to calculate which elements may be selected by the viewer in the future course of the hypervideo. Different ways of scheduling elements for pre-fetching and download are possible. One strategy would be to analyze all possible paths from a certain point in the hypervideo and download all elements of the paths up to a certain time limit (like a sliding window, but over different paths). This method downloads whole scenes leaving no elements out, which is bandwidth consuming. It ensures that all elements are cached for future playback which avoids breaks in the video flow. In our tests in previous work [16, 17], a strategy ("PREFETCH_SL") which schedules and downloads only the part of the scene that is needed to start playback while also loading the missing data of a scene wherein no breaks during playback occur, showed good results. The scheduling of the single scenes is done for each layer. The layers are calculated from a certain scene onward with a breadth first search noting the distances to the current scene. An illustration of the PREFETCH_SL schedule for six scenes can be found in Figure 3. Outgoing from scene 1, scenes 2 and 3 can be reached with one step and are therefore on the second level. Scenes 4, 5 and 6 can be reached with two steps are therefore on the third level. Thereby, scene 5 has two ingoing edges, one with a distance of two steps and one with a distance of three steps. Using a breadth first search, the smaller of the two distances is used for further processing.

Different probabilities can be applied to the single paths depending on the previous user behavior or other settings in the hypervideo defined in the metadata. For the creation of a linear schedule of a level of the scene graph, a slightly modified version of the round robin algorithm described by Shreedhar and Varghese [26] is used where the probability of selecting a scene is applied to the quantum size of the queue.

3.2.3 Calculation of the Start Frame for Playback

When the playback schedule for one scene is known, the point in time from which the scene can be played without breaks after downloading a certain amount of data from the download schedule can be calculated. Doing that, we first calculate the download time $t_{download}$ for the given schedule. Next, the elements of the schedule are downloaded step-by-step. The remaining time for the download $t_{download}$ is reduced with every downloaded element. In parallel, the already downloaded frames are counted and we calculate how much time of the scene can be played back $t_{playback}$. We then compare both values, and if $t_{playback} > t_{download}$, the playback can be started.

3.3 Pre-fetching - Realization in Browsers

The theoretical background and calculations on the definitions described in Section 3.2 help to understand the connections between different algorithms for pre-fetch, download scheduling, cache management, and deletion from the cache. Cache management and deletion from the cache will not be examined in more detail in this work, because it is mainly executed by the Web browser and cannot be influenced by our implementation. The results in [16, 17] showed that the deletion of elements from the cache only has a greater influence in the cycle pattern but does not affect the overall performance significantly.

Trying to "translate" the algorithms and strategies from the simulation framework into a real-world player in a browser, we were facing the following problems:

- Frames could not be downloaded one-by-one, neither in HTML5 nor in the MSE.
- The creation of linear schedules was not possible in HTML5.
- A linear scheduling of a sub-graph was not possible in HTML5 due to the aforementioned limitations.
- The calculation of the start frame for playback could not be implemented on a frame basis.

Most of the problems resulted from the fact that it is not possible to apply such a fine granularity of pre-fetch and

download scheduling as used in the simulation framework to currently existing web technologies. In the following, we will describe how the algorithms introduced in Section 3.2 can be modified and realized with HTML5 [36] and the MSE [37, 38]. See Table 1 for an overview and comparison.

3.3.1 Implementation in HTML5

Using the standard HTML5 video tag, different attributes can be set and properties can be queried. The relevant ones are:

- `preload (auto|metadata|none)`: Setting `preload` to "`none`", no pre-loading is performed. Using "`metadata`", only the metadata of the video but not the video itself is loaded. The standard setting is "`auto`", it triggers an optimistic pre-fetching of the whole video.
- `buffered`: The `buffered` property contains the time ranges (start and end times) of parts of the video that are already loaded into the browser cache.
- `readystate`: The `readystate` property indicates the state of the video:
  - `HAVE_NOTHING`: No data were loaded so far.
  - `HAVE_METADATA`: The metadata are loaded.
  - `HAVE_CURRENT_DATA`: The currently needed frame was loaded.
  - `HAVE_FUTURE_DATA`: Future frames were loaded.
  - `HAVE_ENOUGH_DATA`: Enough data were loaded to be able to most likely play the video without any breaks.

The initial version of the HTML5 hypervideo player did not analyze the video in any manner. Therefore, the player was extended with two new components, the "fork analyzer" and the "pre-fetcher". The fork analyzer analyzes the scene graph of the hypervideo and determines which scenes may be played in the future course of playback. It also identifies how many layers are between the current scene and possible future scenes. When a viewer selects one of the scenes, the pre-fetcher determines which scenes will be scheduled for pre-load, performs the pre-load, and calculates the time from which the scene can be played without breaks.

Accordingly, the player first gets the scene graph from the metadata. When a scene is selected, the fork analyzer determines the next possible scenes. Then the scene is analyzed by the pre-fetcher and the download from the server starts. When enough data are available (as calculated by the pre-fetcher), the playback starts. If there is still bandwidth available after all elements of the scene are downloaded, the pre-loading of future scenes is started.

However, some adjustments of the algorithms described in Section 3.2 are necessary:

- **Scheduling a scene**: Using HTML5, it is not possible to implement a linear schedule for a scene. For pre-fetching a scene, it is only possible to chose between the `preload` options `auto`, `metadata`, or `none`. A higher granularity is not possible. Furthermore, it is not possible to pause the buffering of a video. For that reason, additional information is downloaded in parallel with the main video when the (already buffered) main video reaches the point where the additional information should be displayed. Thereby the guarantee is lost that annotations are pre-fetched for display and can be displayed immediately. The main video prefetch as well as the handling of parallel downloads may be different using different web browsers.
- **Scheduling a sub-graph**: It is possible to determine the levels of the scene graph. Accordingly a pre-fetch of selected scenes is realizable. However, as described for scheduling a scene, the same restrictions apply for scheduling a sub-graph. This results in parallel downloads for scenes of one level. Furthermore, the prioritization of scenes can not be realized. In order to adapt to the viewer behavior, we pre-fetch all scenes with a priority higher than the average priority. If all scenes have the same priority, all are pre-fetched.
- **Calculation of the start frame for playback**: An approximation of the calculations described in Section 3.2 can be used to determine the point in time for starting the playback. However, different browsers handle the pre-fetch of elements differently. Accordingly, the calculated point in time to start playback may never be reached and thus the playback of the video will never be started. A work-around is checking the `readystate` and waiting for the event `canplaythrough`. This does not ensure the availability of the annotations and may result in a displaying delay. Like the handling of the pre-fetch of elements, the browsers are also handling the throwing of events differently.

3.3.2 Implementation with Media Source Extensions

The Media Source Extensions (MSE) allow a smaller deviation from the algorithms described in Section 3.2 in contrast to the previously described implementation of the buffering in HTML5. Currently, the MSE are in the state of a W3C Candidate Recommendation [37] or rather a W3C Editor's draft [38]. The two most important features of this future standard are client sided buffering and adaptive streaming via HTTP (DASH). Client sided buffering allows the pre-fetching of parts of the video, which can than be added to the video tag in the form of byte streams. For the realization of our algorithms, a separation of the video into downloadable "clusters" is necessary. "Clusters" of frames are a specified set of consecutive frames of a single video file - or with regard to hypervideo in a single scene. While the WebM container already provides clusters, MP4 files need to be modified because the internal structure of the container must be adapted. The buffering is then done by MSE. First a new MediaSource object is created which is then used as a source for the HTML5 video tag. A source buffer is added to the MediaSource object. Then video clusters are added and the buffer is filled.

In order to extend the existing HTML5 hypervideo player with a pre-fetching strategy using MSE, three new components ("scene analyzer", "fork analyzer", and "pre-fetcher") were added. The scene analyzer calculates linear download schedules and the overall download volume for individual scenes. The fork analyzer calculates the distances to possible future scenes. It calculates a complete download schedule for the current situation. The pre-fetcher calculates the start point for the playback and applies the download schedule calculated by the fork analyzer.

Accordingly, first schedules for scenes are created, then the structure of the scene graph is fetched as well as information about the different elements. After selecting a scene, possible next scenes are analyzed and the schedule is refreshed. Then the schedule (of the current and possible future scenes) is applied and the elements are downloaded

Table 1: Comparison of the implementations.

	Status quo HTML5		MSE		
			immediately	***wait***	***wait no break***
Calculate start frame for playback	start playback depending on implementation in browser	approximated calculation as described in Section 3.2.3 in combination with checking the ready-state for the `canplaythrough` event (implementation browser dependent)	similar to implementation in HTML5	as described in Section 3.2.3 but based on clusters instead of frames	more optimistic than the "wait no break" strategy but may lead to breaks; differentiates between main video and annotations; takes into account that downloaded clusters lead to more available playback time
Schedule scene	no prefetch implemented	preload options: `auto`, `meta`, `none`; additional information downloaded parallel with main video; no guarantee for immediate display of annotations	linear scheduling of clusters (instead of frames) of scene as described in Section 3.2.1 - more pessimistic calculation of start frame		
Schedule sub-graph	no prefetch implemented	parallel download of scenes on one level of the graph; if same prio, all scenes downloaded parallel; if one scene has higher prio, only this scene downloaded	scheduling based on clusters, not frames - priorities applied to clusters in scenes, not frames		

from the HTTP server. Playback starts when enough elements are in the buffer.

As described for the implementation in HTML5, MSE also needs adjustments to the algorithms described in Section 3.2:

- **Scheduling a scene**: Due to the fact that a separation of the video into clusters is possible, a linear download schedule can be calculated for a scene. The schedule is calculated as described in Section 3.2.1 with the difference that the scheduling is not based on single frames but on available clusters. This may lead to a little more pessimistic calculation of the start time as the calculations based on frames.
- **Scheduling a sub-graph**: Like the scheduling of scenes, the scheduling of parts of the graph is also possible, again with the limitation on clusters instead of individual frames. It is possible to realize the prioritization of scenes by adding more clusters to a schedule for higher prioritized scenes.
- **Calculate start frame for playback**: We introduce three different ways to calculate the start frame:
  - **MSE immediately**: This strategy is similar to the one described for the HTML5 implementation.
  - **MSE wait no break**: This strategy is implemented as described in Section 3.2.3 with the limitation of using clusters instead of individual frames.
  - **MSE wait**: The "MSE wait no break" strategy may result in comparably long waiting times. For that reason we implemented a further variant to calculate the start frame. It cannot guarantee a playback without breaks, but limits them to a minimum with significantly lower beginning scene waiting times. This strategy differentiates between the download of additional information and the main video of a scene and takes into account the fact that downloaded clusters of the video result in further available playback time.

4. EVALUATION

In the following, we present our evaluation results. To measure the quality of our algorithms, we defined metrics for the analysis of the quality of the algorithms. We test or algorithms using patterns which occur in scene graphs. We present and discuss our results at the end of this section.

4.1 Metrics

In order to determine the quality of our algorithms, we use five metrics, namely:

- ***Wait start***: sum of waiting times from selecting a scene until the playback starts
- ***Wait playback***: sum of waiting times during playback after breaks if needed elements are not in buffer
- ***Wait total***: *wait start + wait playback*
- ***Breaks***: occurrence of a break in the video flow for more than 500 ms
- ***Download volume***: overall data volume, sum of file sizes of all downloaded files

For a more formal definition of the metrics, please refer to [16, 17]. The metrics commonly used for web caching (see [25] for an overview) have only little informative value in the area of hypervideos.

4.2 Patterns

Dealing with hypervideo, two different sub-patterns exist for each pattern: the structure pattern (black colored arrows in our figures) and the user-path pattern (blue colored dashed arrows in our figures). The structure pattern describes the connections between the video scenes and the additional information representing the edges of the scene graph. The user-path pattern is the sequence of scenes from the structure pattern describing the path that was selected through the scene graph by the viewer.

Testing on patterns instead of complete hypervideos has the advantage of measuring relevant data without having to test on a large number of different hypervideos, which

Table 2: Settings for video and additional information for the patterns.

Scene name	dur. [sec]	size [MB]	add. info	display [sec]	sizes [MB]
S01	10	0.54			
S02	32	29.8	A02	10-20	6.37/7.18/5.68/5.36
S03	60	60.3	A03	30-40	8.06/6.83/9.57
S04	41	28.5			
S05	45	56.7	A05	00-10	7.40/5.51
S06	4	2.1	A06	00-04	2.32
S07	30	15.9	A07	00-10	1.21
S08	42	44.8	A08	00-10	1.39
S09	60	53.1	A09	10-20	3.24/5.92
S10	59	53.7	A10	20-30	2.24/2.80/2.76/2.62
S11	37	72.4	A11	10-20	3.13/1.35
S12	46	36.8			
S13	60	36.7	A13	30-40	4.96/6.51
S14	61	55.6	A14	20-30	3.88/5.48/4.71

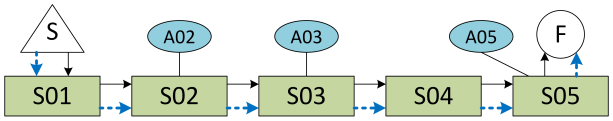

Figure 4: Sequence pattern with five scenes and three blocks of additional information.

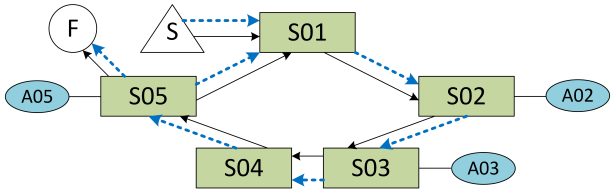

Figure 5: Cycle pattern with five scenes and three blocks of additional information.

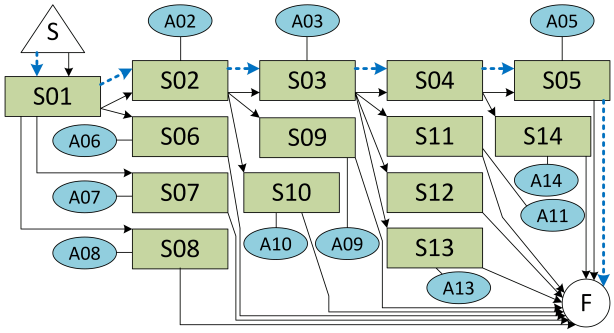

Figure 6: Split pattern with 14 scenes and eleven blocks of additional information.

would be necessary to get reliable results due to their large variance in structure, depending on the usage scenario. All hypervideos contain the same sub-patterns. The test-results for the sub-patterns can then be combined into a recommendation for a specific hypervideo. The patterns used in our evaluation are derived from the analysis of existing scene graphs that were created for different scenarios using our software (see [19, 21, 29] for further descriptions). We also analyzed the work of Bernstein [6] for the suitability of the patterns in hypertext for structures in hypervideos. Basically three patterns reoccur and will therefore be analyzed in detail:

- **Sequence pattern**: The sequence pattern provides a linear series of scenes from the start to the end of the video. Hereby, every scene is watched by the viewer. This results in the same user-pattern as the structure pattern. The sequence pattern used in the tests in this work has five video scenes. Three of them have annotations. An illustration of this pattern can be found in Figure 4. The exact durations of scenes, files sizes, and display times of annotations can be found in Table 2.
- **Cycle pattern**: The cycle pattern has two or more scenes which can be watched more than once. Every scene is watched by the viewer. At the last scene of the cycle, the viewer can decide if she/he wants to watch the scenes one more time. Hereby, the user-pattern may even be larger than the structure pattern. The cycle pattern used in our tests contains the same scenes and annotations as the sequence pattern (see Figure 5). The exact durations of scenes, files sizes, and display times of annotations can be found in Table 2. The cycle is taken three times by the viewer in our tests. It gives hints on how the cache is managed internally.
- **Split pattern**: The scenes in the split pattern have at least two successor scenes of which one can be selected by the viewer. The viewer usually only watches a small subset of the scenes of the structure pattern, resulting in a much smaller user-path pattern than the structure pattern. The split pattern in our tests contains 14 scenes whereof eleven have annotations. Only five scenes are selected, see Figure 6. The exact durations of scenes, files sizes, and display times of annotations can be found in Table 2. We apply different probabilities to the paths in this pattern resulting in three different sub-tests. One test is performed with equal probabilities for all scenes at a fork in the scene graph. A test with very low probabilities for the selected user-path is performed with the following settings: S1 $\rightarrow$ S2: 7%, S2 $\rightarrow$ S3: 5%, S3 $\rightarrow$ S4: 5%, and S4 $\rightarrow$ S5: 10%. The test with high probabilities for the selected user-path uses these settings: S1 $\rightarrow$ S2: 70%, S2 $\rightarrow$ S3: 80%, S3 $\rightarrow$ S4: 85%, and S4 $\rightarrow$ S5: 90%. This pattern gives hints about the quality of the pre-fetching algorithms.

To provide a comparability between the patterns, the same five scenes are always selected for playback, S01 to S05, resulting in a playback time of 188 sec. This playback time is enough because the relative results for the metrics that emerge from the scene changes do not change if they are measured for longer playback paths. These scenes have the same additional information in each scenario resulting in the same minimum download volume for each test. The videos were VP8 encoded WebM videos with a 720p HD resolution. All additional information were JPG images.

4.3 Test Environment

Each pattern is tested with the bandwidths 11 Mbit/s,

Table 3: Statistics: split pattern, MSE immediately, bandwidth 11 Mbit/s, 50 runs

	Wait start [ms]	Wait playback [ms]	Breaks [count]	Download volume [Mbit/s]
Minimum	227	26771	4	292.47
Maximum	359	27356	4	293
Average	261.14	27048.82	4	292.74
SD	20.07	118.64	0	0.13
1st quart.	249.5	26973.25	4	292.63
2nd quart.	260	27037	4	292.72
3rd quart.	268	27115	4	292.86

13.5 Mbit/s, and 16 Mbit/s. These bandwidths were derived from currently available exemplary average connection speeds throughout the world [5]. 16 Mbit/s represent an average speed available in Norway or Switzerland, 13.5 Mbit/s is available in Denmark or the UK, and 11 Mbit/s is, on average, reached in Slovakia or Hungary. To provide these bandwidths, we used a local web server and the maximum bandwidth was limited using the NetLimiter tool [14] (version 4.0.13.0) at client side. All tests were performed in an automated way, with the waiting time for selecting the next scene set to three seconds. The values were logged for the evaluation. All test runs were performed on a Windows 8.1 computer using the Chrome Browser (version 42.0.2311.152 m) and a browser cache size setting of 209.71 MB. With a minimum download volume of 239 MB for a user-path, not all elements fitted into the cache accordingly. The files used in the evaluation can be requested from the authors of this paper.

4.4 Statistics

Each of the 75 test runs (5 pattern * 3 download speeds * 5 algorithm settings) is performed in real time, resulting in a playback time of 188 seconds. Adding waiting times, download times, and times for test setup we wanted to minimize the time needed for the tests. Accordingly, we performed a pre-test of 50 test runs in one representative setting (split pattern, average probabilities, MSE immediately, bandwidth 11 Mbit/s) to evaluate the variations of the result values. The results showed that the variations were very small to non-existent (see Table 3 for the exact values). The difference between the first and the third quartile for the waiting time before playback is 17.5 ms which is less than the display duration of a frame. The difference between the first and the third quartile for the waiting time during playback is 0.14 sec, which is less than 0.5 sec (the duration for recognizing a break). The number of breaks is the same for all test runs. The difference between the first and the third quartile for the download volume is 0.22 MB which is one sixth of the smallest used annotation size. Having shown that the variances between the different test runs are very small, we concluded that one run per test setting is enough to get meaningful results (using the same preconditions in the test environment as for the statistics test).

4.5 Breaks During Playback

Taking a look at the number of breaks during scenes, it can be stated that the status quo has by far the most breaks for each bandwidth, pattern, and on average ($\overline{x} = 8.1$ breaks).

Table 4: Number of playback breaks (0-0.9 → green, 1-3.9 → yellow, 4-9.9 → orange, ≥ 10 → red)

Pattern	Bandwidth	status quo	HTML5	MSE imm.	MSE wait	MSE w.n.b.	$\overline{x}$
Sequence	11	12	5	1	0	0	**3.6**
	13.5	8	1	1	0	0	**2.0**
	16	4	0	0	0	0	**0.8**
	$\overline{x}$	8	2	0.67	0	0	**2.1**
Cycle	11	10	5	2	1	0	**3.6**
	13.5	7	1	0	0	0	**1.6**
	16	6	0	0	0	0	**1.2**
	$\overline{x}$	7.67	2	0.67	0.33	0	**2.1**
Split high	11	13	6	2	1	0	**4.4**
	13.5	7	1	1	1	0	**2.0**
	16	5	0	0	0	0	**1.0**
	$\overline{x}$	8.33	2.33	1	0.67	0	**2.5**
Split avg	11	12	9	4	2	0	**5.4**
	13.5	7	6	2	1	0	**3.2**
	16	6	5	1	1	0	**2.6**
	$\overline{x}$	8.33	6.67	2.33	1.33	0	**3.7**
Split low	11	13	13	2	1	0	**5.8**
	13.5	7	7	2	1	0	**3.4**
	16	4	4	1	1	0	**2.0**
	$\overline{x}$	8	8	1.67	1	0	**3.7**
	$\overline{x}$	**8.1**	**4.2**	**1.3**	**0.7**	**0.0**	

The best results are achieved for the "MSE wait no break" implementation which has no breaks in any of the test cases. Slightly worse results are achieved by the "MSE wait" implementation with a maximum of two breaks during playback on average and for all patterns, except the sequence pattern where both implementations show the same results. A maximum of four breaks is achieved by the "MSE immediately" implementation which achieves on average slightly worse results than the "MSE wait" implementation ($\overline{x} = 1.3$ breaks vs. $\overline{x} = 0.7$ breaks). The HTML5 implementation shows worse results than all of the MSE implementations, but better results than the status quo except for the split pattern with low probabilities on the user path, where both implementations show the same number of pauses. Using the pre-fetching in HTML5 can reduce the breaks by half ($\overline{x} = 4.2$ breaks) compared to the status quo without any pre-fetch. The exact results are given in Table 4.

4.6 Waiting Times

Taking a look at the waiting times in the different patterns, we distinguish between the waiting time at the beginning of scenes, the time to wait after a break during a scene, and the sum of both times. The lowest waiting time at the beginning of scenes was achieved by the improved HTML5 implementation for each of the patterns as well as on average. Similarly low beginning scene waiting times were also measured using the status quo and the "MSE immediately" strategies. The lowest waiting time during scenes was reached by the "MSE wait no break" strategy for all patterns as well as on average. The lowest overall waiting time

Table 5: Waiting times at different bandwidths, at the beginning and during scenes (0-3 → green, 3.1-6 → yellow, 6.1-12 → orange, ≥ 12.1 → red), as well as summarized waiting times (0-6 → green, 6.1-12 → yellow, 12.1-24 → orange, ≥ 24.1 → red).

Pattern	Bandw.	status quo			HTML5			MSE immediately			MSE wait			MSE wait no break			
		start	during	sum	start	during	sum	start	during	sum	start	during	sum	start	during	sum	$\overline{x}(sum)$
Sequence	11	0.3	42.5	42.8	0.1	10.3	10.3	0.3	7.1	7.4	17.8	1.2	19.0	31.6	0.6	32.3	**22.4**
	13.5	0.3	22.6	22.9	0.1	1.9	1.9	0.2	1.7	2.0	0.9	1.1	2.1	10.4	0.6	11.0	**8.0**
	16	0.3	11.3	11.6	0.1	0.6	0.7	0.3	1.1	1.3	0.8	0.7	1.5	3.1	0.7	3.8	**3.8**
	$\overline{x}$	0.3	25.5	25.8	0.1	4.3	4.3	0.3	3.3	3.6	6.5	1.0	7.5	15.0	0.6	15.7	**11.4**
Cycle	11	0.8	44.8	45.8	0.1	14.1	14.3	0.8	14.7	15.5	35.1	3.2	38.3	69.2	2.0	71.2	**37.0**
	13.5	0.8	27.2	27.9	0.1	3.5	3.7	0.8	4.1	4.8	2.8	2.4	5.3	22.6	1.9	24.5	**13.2**
	16	0.8	16.6	17.4	0.1	2.1	2.3	0.9	3.5	4.3	2.3	2.2	4.52	7.9	1.9	9.8	**7.7**
	$\overline{x}$	0.8	29.5	30.4	0.1	6.6	6.8	0.8	7.4	8.2	13.4	2.6	16.0	33.2	1.9	35.2	**19.3**
Split high	11	0.3	41.6	41.9	0.1	10.5	10.6	0.3	16.4	16.7	29.9	1.5	31.4	45.6	0.7	46.3	**29.4**
	13.5	0.3	24.6	24.9	0.1	2.0	2.1	0.3	6.2	6.5	4.9	3.0	7.9	29.0	0.6	29.6	**14.2**
	16	0.3	11.4	11.7	0.1	0.6	0.7	0.2	1.8	2.0	0.8	1.2	2.0	12.9	0.7	13.5	**6.0**
	$\overline{x}$	0.3	25.9	26.2	0.1	4.4	4.5	0.3	8.1	8.4	11.9	1.9	13.8	29.2	0.7	29.8	16.5
Split avg.	11	0.3	40.9	41.2	0.0	30.0	30.0	0.3	27.0	27.3	37.6	2.0	39.6	71.7	0.6	72.3	**42.1**
	13.5	0.3	24.0	24.4	0.0	15.1	15.1	0.3	14.2	14.4	18.7	3.2	22.0	47.4	0.7	48.0	**24.8**
	16	0.3	15.4	15.7	0.1	8.9	8.9	0.3	9.0	9.3	7.6	1.7	9.2	24.3	0.7	25.0	**13.6**
	$\overline{x}$	0.3	26.8	27.1	0.0	18.0	18.0	0.3	16.7	17.0	21.3	2.3	23.6	47.8	0.7	48.4	**26.8**
Split low	11	0.3	42.0	42.3	0.3	42.3	42.6	0.3	28.1	28.4	41.3	1.7	43.0	78.2	0.6	78.8	**47.0**
	13.5	0.3	24.0	24.3	0.3	24.6	24.9	0.3	16.0	16.3	20.5	3.3	23.8	53.1	0.7	53.8	**28.6**
	16	0.3	11.2	11.6	0.3	12.6	12.9	0.3	10.9	11.2	9.5	1.7	11.1	33.9	0.6	34.5	**16.3**
	$\overline{x}$	0.3	25.7	26.1	0.3	26.5	26.8	0.3	18.3	18.6	23.8	2.2	26.0	55.1	0.6	55.7	**30.6**
	$\overline{x}$	**0.4**	**26.7**	**27.1**	**0.1**	**11.9**	**12.1**	**0.4**	**10.8**	**11.2**	**15.4**	**2.0**	**17.4**	**36.1**	**0.9**	**37.0**	

depends on the underlying structure pattern. The "MSE immediately" implementation achieves the best results for the sequence pattern, the split pattern with average or low probabilities on the user path, as well as on average. The HTML5 implementation performs better for the Cycle pattern and especially for the split pattern with high probabilities on the user path.

The highest beginning scene waiting times are reached by the "MSE wait no break" strategy for all patterns individually and averaged. This results from the pre-fetch behavior and the calculation of the start frame which makes sure that no breaks occur during playback. Most of the content loaded into the cache results in high to very high beginning scene waiting times. The highest during scenes waiting times are achieved by the status quo on average and for all patterns except for the split pattern with low probabilities on the user path. The highest overall waiting times are reached by the "MSE wait no break" implementation on average and for all patterns except for the sequence pattern, where the status quo achieves longer waiting times.

To summarize these findings, it can be noted that the HTML5 or the "MSE immediately" implementations achieve the shortest overall waiting times, but show longer waiting times during the scenes, especially for the split pattern with average or low probabilities on the user path. The "MSE wait" implementation has slightly longer overall waiting times, where only a very small amount of time has to be waited during playback. The status quo has the shortest beginning scene waiting times, but very long during scene waiting times. "MSE wait no break" represents the opposite behavior, having very long beginning scene waiting times and very short during scene waiting times. Another finding is that the number of breaks correlates with the waiting times after a break. Accordingly, it can be noted that each break leads to a certain waiting time. An overview of all results is given in Table 5.

4.7 Overall Download Volume

The overall download volume is always the lowest for the status quo implementation on average and for all patterns, because no pre-fetching is done and all elements are downloaded when they are needed. The minimum download volume for all patterns is 239 MB, which is also the size of all scenes and annotations together for the sequence and the cycle pattern. The size of all elements for the split pattern is about three times as high (663.42 MB). Equal results are achieved by the HTML5 implementation with pre-fetch for the sequence, the cycle, and the split pattern with high priorities for the user path. All MSE strategies achieve the minimum available download volume of 239 MB for the sequence pattern. The download volumes are higher for each of the MSE implementations for the split pattern, while the "MSE immediately" implementation requires less volume ($\overline{x} = 358.7MB$) than the "MSE wait" ($\overline{x} = 377.8MB$) and the "MSE wait no break" ($\overline{x} = 384.5MB$) implementation. An overview of all results is given in Table 6.

The values for the cycle pattern are much higher for the MSE implementations compared to the other patterns. The available buffer size in the tests is 209 MB. The amount of data of one cycle is 239 MB, which is slightly higher than the overall available buffer size. One could assume that all implementations result in a higher download vol-

Table 6: Overall download volume in MB (239 MB → green, 239.1-318.7 MB → yellow, 318.8-398.3 MB → orange, ≥ 398.4 MB → red)

Pattern	Bandwidth	status quo	HTML5	MSE imm.	MSE wait	MSE w.n.b.	$\overline{x}$
Sequence	11	239	239	239	239	239	**239.0**
	13.5	239	239	239	239	239	**239.0**
	16	239	239	239	239	239	**239.0**
	$\overline{x}$	239.0	239.0	239.0	239.0	239.0	**239.0**
Cycle	11	239	239	570	662	664	**474.8**
	13.5	239	239	654	672	541	**469.0**
	16	239	239	570	672	672	**478.4**
	$\overline{x}$	239.0	239.0	598.0	668.7	625.7	**474.1**
Split high	11	239	239	267	281	289	**263.0**
	13.5	239	239	283	283	324	**273.6**
	16	239	239	333	333	346	**298.0**
	$\overline{x}$	239.0	239.0	294.3	299.0	319.7	**278.2**
Split avg.	11	239	272	293	309	336	**289.8**
	13.5	239	285	337	346	367	**314.8**
	16	239	296	363	363	386	**329.4**
	$\overline{x}$	239.0	284.3	331.0	339.36	363.0	**311.3**
Split low	11	239	272	294	313	345	**292.6**
	13.5	239	272	330	346	376	**312.6**
	16	239	272	370	370	404	**331.0**
	$\overline{x}$	239.0	272.0	331.3	343.0	375.0	**312.1**
	$\overline{x}$	**239.0**	**254.7**	**358.7**	**377.8**	**384.5**	

ume, because the buffer is eventually full and has to be deleted. But the status quo and the HTML5 implementation only need the minimum download volume. This can be explained by the fact that the Chrome Browser has a media cache in addition to the standard cache. A large amount of the downloaded data is moved into this media cache and not kept in the standard browser cache, thus retransmissions are avoided. Using MSE, the downloaded clusters are not recognized as media files and therefore stored in the standard browser cache. The data of one cycle exceeds the cache size, resulting in a deleting of the elements from the cache and a repeated download during the next cycle. Knowing this for the Chrome Browser, we also examined the caching behavior for other available browsers for the cycle pattern. The Opera Browser (version 31.0) shows the same behavior as the Chrome Browser. Internet Explorer (version 11.0.9.600.17905) has no dedicated media cache. Its standard cache size is 250 MB, so the data of the whole cycle fit into the cache and nothing has to be deleted. The Firefox Browser (version 39.0) also only has one cache with a standard cache size of 367 MB. Tests showed that not all data are stored in the cache, which resulted in higher values for the download volume for all implementations using Firefox.

5. GUIDELINES

From the findings of the previous section and the results in [16, 17], we can derive the following guidelines for the implementation of hypervideo web players with HTML5 or MSE:

- ***Minimization of breaks:*** Use the "MSE wait" implementation as described in this paper. It does not avoid breaks at all costs but keeps the waiting time at the beginning of scenes as low as possible.
- ***Minimization of waiting times before scenes:*** Use "MSE immediately" or HTML5. The "MSE immediately" implementation provides a solution for less breaks during scenes and the HTML5 implementation results in a lower download volume.
- ***Minimization of the download volume:*** Use the HTML5 implementation. It needs a little more volume than the "status quo", but provides much better results for waiting times and breaks during scenes.
- ***Hypervideo with mainly sequences or many splits of paths:*** Use HTML5 or "MSE immediately".
- ***Hypervideo with many circles:*** Use the HTML5 implementation because it performs better if the browser provides a separate media cache (see Section 4.7).

6. CONCLUSIONS

In this work we presented hypervideo download and cache management algorithms implemented in HTML5 and MSE. We compared our new implementations to the status quo of an HTML5 hypervideo player. We tested the implementations with three different patterns (sequence, cycle, and split pattern), at three different bandwidths (11 Mbit/s, 13.5 Mbit/s, and 16 Mbit/s), and one set of video scenes and annotations. We tested larger file sizes with higher bandwidth in our previous work [16, 17] and discovered that if file size and bandwidth increase linearly the effects of our strategies stay the same.

The results show that significantly lower overall waiting times can be achieved if either HTML5 pre-fetching or MSE is used. We implemented three different versions of the algorithms for MSE ("MSE immediately", "MSE wait", and "MSE wait no break") that balance the beginning and during scene waiting times differently. "MSE immediately" and HTML5 with pre-fetching show similar results in many test settings. The overall waiting time of the "MSE wait" implementation is higher than for the "MSE immediately" and the HTML5 with pre-fetching implementations. It shows longer beginning scene waiting times, but results in no, or only very few, breaks. With much shorter overall waiting times, the "MSE wait" implementation is preferable to the "MSE wait no break" implementation. Overall download volume strongly depends on the browser used and its internal cache management. It is especially relevant when the hypervideo has many cycles which may be viewed more than once. Due to the caching behavior of the browsers using HTML5, retransmissions are avoided.

In future work, we want to test our algorithms in hypervideos that contain a table of contents and a keyword search function. In order to be able to deal with random jumps in a hypervideo's structure we want to see how the algorithms described in this work need to be improved.

7. ACKNOWLEDGMENTS

This work was funded by the Bundesministerium für Bildung und Forschung (German Federal Ministry of Education and Research) (BMBF) under project number 03V0633.

8. REFERENCES

[1] Svg smil animation. Website (accessed October 17, 2015), 2015. http://caniuse.com/feat=svg-smil.

[2] Adobe Systems Incorporated. Actionscript 3.0 reference for the adobe flash platform. Website (accessed October 12, 2015), 2015. http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/index.html.

[3] Adobe Systems Incorporated. Adobe flash professional cc. Website (accessed October 12, 2015), 2015. http://www.adobe.com/products/flash.html.

[4] Adobe Systems Software Ireland Ltd. Adobe flash platform runtimes. statistics : Pc penetration. Website (accessed October 12, 2015), 2011. http://www.adobe.com/de/products/flashplatformruntimes/statistics.html.

[5] Akamai Technologies. akamai's [state of the internet] - q3 2015 report. Website (accessed January 13, 2016), 2015. https://www.akamai.com/us/en/multimedia/documents/report/q3-2015-soti-connectivity-final.pdf.

[6] M. Bernstein. Patterns of hypertext. In *Proceedings of the Ninth ACM Conference on Hypertext and Hypermedia: Links, Objects, Time and Space - structure in Hypermedia Systems: Links, Objects, Time and Space - structure in Hypermedia Systems*, HYPERTEXT '98, pages 21–29, New York, NY, USA, 1998. ACM.

[7] Cisco. Cisco visual networking index: Global mobile data traffic forecast update 2014-2019 white paper. Visual networking index (vni), white paper, CISCO, February 2015.

[8] Cisco. Vni forecast highlights. Website (accessed October 13, 2015), 2015. http://www.cisco.com/web/solutions/sp/vni/vni_forecast_highlights/index.html.

[9] Codevise Solutions Limited. Pageflow. das tool für multimediales storytelling. Website (accessed October 12, 2015), 2015. http://pageflow.io/de.

[10] Z. Fei, M. Ammar, I. Kamel, and S. Mukherjee. Providing interactive functions through active client-buffer management in partitioned video multicast vod systems. In L. Rizzo and S. Fdida, editors, *Networked Group Communication*, volume 1736 of *Lecture Notes in Computer Science*, pages 152–169. Springer Berlin Heidelberg, 1999.

[11] R. Grigoras, V. Charvillat, and M. Douze. Optimizing hypervideo navigation using a markov decision process approach. In *Proceedings of the Tenth ACM International Conference on Multimedia*, MULTIMEDIA '02, pages 39–48, New York, NY, USA, 2002. ACM.

[12] Honkytonk Films. Klynt. Website (accessed October 12, 2015), 2015. http://www.klynt.net/.

[13] R. Laraspata, D. Striccoli, and P. Camarda. A scheduling algorithm for interactive video streaming in umts networks. In *Computers and Communications (ISCC), 2010 IEEE Symposium on*, pages 997–1002, June 2010.

[14] Locktime Software. Welcome to netlimiter. netlimiter is an ultimate internet traffic control and monitoring tool designed for windows. Website (accessed August 30, 2015), 2015. http://www.netlimiter.com/.

[15] mc-quadrat. Linius. das storytelling-tool. Website (accessed October 12, 2015), 2015. http://linius-storytelling.de/.

[16] B. Meixner. *Annotated Interactive Non-linear Video - Software Suite, Download and Cache Management*. Phd thesis, Universität Passau, 2014.

[17] B. Meixner. A pattern-based evaluation of download and cache management algorithms for annotated interactive non-linear videos. *Multimedia Systems*, pages 1–35, 2016.

[18] B. Meixner and H. Kosch. Interactive non-linear video: Definition and xml structure. In *Proceedings of the 2012 ACM Symposium on Document Engineering*, DocEng '12, pages 49–58, New York, NY, USA, 2012. ACM.

[19] B. Meixner, B. Siegel, G. Hölbling, F. Lehner, and H. Kosch. Siva suite: Authoring system and player for interactive non-linear videos. In *Proceedings of the International Conference on Multimedia*, MM '10, pages 1563–1566, New York, NY, USA, 2010. ACM.

[20] B. Meixner, B. Siegel, P. Schultes, F. Lehner, and H. Kosch. An html5 player for interactive non-linear video with time-based collaborative annotations. In *Proceedings of International Conference on Advances in Mobile Computing & Multimedia*, MoMM '13, pages 490:490–490:499, New York, NY, USA, 2013. ACM.

[21] B. Meixner, K. Tonndorf, S. John, C. Handschigl, K. Hofmann, and M. Granitzer. A multimedia help system for a medical scenario in a rehabilitation clinic. In *Proceedings of the 14th International Conference on Knowledge Technologies and Data-driven Business*, i-KNOW '14, pages 25:1–25:8, New York, NY, USA, 2014. ACM.

[22] Microsoft. Microsoft silverlight. Website (accessed October 12, 2015), 2015. http://www.microsoft.com/silverlight/.

[23] Microsoft Corporation. Microsoft silverlight release history. Website (accessed October 12, 2015), 2015. http://www.microsoft.com/getsilverlight/locale/en-us/html/Microsoft%20Silverlight%20Release%20History.htm.

[24] Mozilla Developer Network. <video>. Website (accessed October 12, 2015), 2015. https://developer.mozilla.org/en-US/docs/Web/HTML/Element/video.

[25] S. Podlipnig and L. Böszörmenyi. A survey of web cache replacement strategies. *ACM Comput. Surv.*, 35(4):374–398, Dec. 2003.

[26] M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round robin. *SIGCOMM Comput. Commun. Rev.*, 25(4):231–242, Oct. 1995.

[27] M. Silverman. The history of html5. Website (accessed October 12, 2015), 2012. http://mashable.com/2012/07/17/history-html5.

[28] Stack Exchange Inc. (User: AndreaC). Code::trends - a visualization of the stackoverflow dataset. Website (accessed October 12, 2015), 2012. http://stackapps.com/questions/3509/codetrends-a-visualization-of-the-stackoverflow-dataset.

[29] K. Tonndorf, T. Knieper, B. Meixner, H. Kosch, and F. Lehner. Challenges in creating multimedia instructions for support systems and dynamic

problem-solving. In *Proceedings of the 12th International Conference on Knowledge Management and Knowledge Technologies*, i-KNOW '12, pages 33:1–33:4, New York, NY, USA, 2012. ACM.

[30] E. Trautman. Rip flash: Why html5 will finally take over video and the web this year. Website (accessed October 12, 2015), 2014. http://thenextweb.com/dd/2014/04/19/rip-flash-html5-will-take-video-web-year/.

[31] W3C. Smil 2.0 testimonials. Website (accessed October 17, 2015). http://www.w3.org/2001/08/smil2-testimonial.

[32] W3C. Synchronized multimedia integration language. w3c working draft 09-november-97. Website (accessed October 17, 2015), 1997. http://www.w3.org/TR/WD-smil-971109.

[33] W3C. Smil animation. w3c recommendation 04-september-2001. Website (accessed October 17, 2015), 2001. http://www.w3.org/TR/2001/REC-smil-animation-20010904/.

[34] W3C. Html 5. a vocabulary and associated apis for html and xhtml. w3c working draft 22 january 2008. Website (accessed October 12, 2015), 2008. http://www.w3.org/TR/2008/WD-html5-20080122/.

[35] W3C. Synchronized multimedia integration language (smil 3.0). w3c recommendation 01 december 2008. Website (accessed October 17, 2015), 2008. http://www.w3.org/TR/SMIL3/.

[36] W3C. Html5. a vocabulary and associated apis for html and xhtml. w3c recommendation 28 october 2014. Website (accessed October 12, 2015), 2015. http://www.w3.org/TR/html5/.

[37] W3C. Media source extensions. w3c candidate recommendation 31 march 2015. Website (accessed October 15, 2015), March 2015. http://www.w3.org/TR/media-source/.

[38] W3C. Media source extensions. w3c editor's draft 13 october 2015. Website (accessed October 15, 2015), March 2015. http://w3c.github.io/media-source/.

[39] Wavelength Media. Flash player version history. Website (accessed October 12, 2015), 2015. http://www.mediacollege.com/adobe/flash/player/version/.

[40] WordPress.com. Wordpress.com. Website (accessed October 12, 2015). https://wordpress.com/.

[41] S. Yegulalp. Adobe flash: Insecure, outdated, and here to stay. Website (accessed October 12, 2015), 2014. http://www.infoworld.com/article/2610420/adobe-flash/adobe-flash--insecure--outdated--and-here-to-stay.html.

[42] YouTube Engineering and Developers Blog. Youtube now defaults to html5 <video>. Website (accessed October 12, 2015), 2015. http://youtube-eng.blogspot.de/2015/01/youtube-now-defaults-to-html5_27.html.