Empowering the Creative User: Personalized HTTP-based Adaptive Streaming of Multi-path Nonlinear Video

Vengatanathan Krishnamoorthi[†] Patrik Bergström[†] Niklas Carlsson[†]
Derek Eager[§] Anirban Mahanti[‡] Nahid Shahmehri[†]
[†] Linköping University, Sweden, firstname.lastname@liu.se
[§] University of Saskatchewan, Canada, eager@cs.usask.ca
[‡] NICTA, Australia, anirban.mahanti@nicta.com.au

ABSTRACT

This paper presents the design, implementation, and validation of a novel system that supports streaming and playout of personalized, multi-path, nonlinear video. In contrast to regular video, in which the file content is played sequentially, our design allows multiple nonlinear video sequences of the underlying (linear) video to be stitched together and played in any personalized order, and clients can be provided multiple path choices. The design combines the ideas of HTTP-based adaptive streaming (HAS) and multi-path nonlinear video. Personalization of the content is achieved with the use of a customized metafile, which is downloaded separately from the underlying media and the manifest file that defines the HAS structure. An extension to the user interface allows path choices to be presented to and made by the user. Novel buffer management and prefetching policies are used to ensure seamless uninterrupted playback regardless of client path choices, even under scenarios in which clients defer their choices until the last possible moment. Our solution allows creative home users to easily create their own multi-path nonlinear video, opening the door to an endless possibility of new opportunities and media forms.

Categories and Subject Descriptors

C.2.2 [**Network Protocols**]: Applications; H.5.1 [**Multimedia Information Systems**]: Video

Keywords

HTTP-based adaptive streaming, Multi-path video, Nonlinear video, Seamless playback

1. INTRODUCTION

The Internet and the World-wide Web is continually transforming the way people access information and content. Increasingly, people use the Internet for their daily news and entertainment. This trend is particularly apparent when considering the current volume of video traffic [4] and the quickly growing content catalogues offered to online users. For example, more than 72 hours of new content is uploaded to YouTube each minute.[1]

1.1 Personalized multi-path nonlinear video

The success of user generated content (UGC) sites such as YouTube shows that there is great value in leveraging the creative power of regular home users. One contribution of this paper is that we enable a solution that further empowers creative developers and home users, such that they easily can define personalized multi-path nonlinear videos.

A relatively simple example generalization of a traditional (linear) video to a multi-path video is a movie in which the viewer is able to choose among a variety of possible endings and/or plot sequences. There are already DVD releases of movies with multiple endings (e.g., the DVD edition of the Hollywood movie "Clue"), and it is easy to imagine generalizations to tree or graph structures.

In this work, we consider a *multi-path, nonlinear* form of video in which: (1) non-contiguous fragments of video can be stitched together to form what we term a *nonlinear* video segment (Figure 1(a)), and (2) the video can include *branch points* at which there are multiple choices of which segment to play back next (Figure 1(b)). The path choices can be selected online (while viewing the media) either by the user, or based on information about the user, including the user's previous path choices. In general, our solution allows the creator to define any arbitrary set of paths and path choices through some original video (or a file that is the concatenation of multiple linear video clips, for example).

1.2 HTTP-based adaptive streaming

There is an increasing demand for *personalized* delivery to increasingly *heterogeneous* users (e.g., using mobile devices). To better utilize the available bandwidth and improve the service of each client, many content providers (including Netflix) have begun using HTTP-based Adaptive Streaming (HAS) [1, 2, 13]. In contrast to basic HTTP-based streaming, with HAS the video is encoded into different qualities and the player, at each point in time, adaptively chooses the most suitable encoding based on the current buffer and network conditions. While HAS allows the service quality to be adapted to the individual user's device and network conditions, it does not customize the content itself.

[1]YouTube statistics, `www.youtube.com/yt/press/statistics.html`, March 2013.

FhMN'13, August 16, 2013, Hong Kong, China.

In this paper we introduce the concept of HAS-based multi-path nonlinear video, effectively enabling customization of the content itself. The use of HAS also has the advantage that the different video sequences are easily addressable by the player and the quality of each path can be adapted to match the available bandwidth.

1.3 Contributions

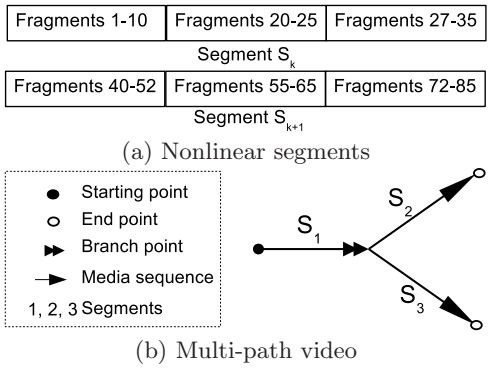
This paper presents the design, implementation, and preliminary validation of HAS-based multi-path nonlinear video streaming. Our design has three novel components.

- **Light-weight personalizable metafile:** To provide developers with maximum possible flexibility, we create a personalizable metafile (defining the multi-path nonlinear structure) that is downloaded separately from the regular media object and the accompanying HAS manifest (that defines quality encodings).
- **Customizable path options and longest-path matching:** Rules associated with each branch-point give the alternatives available to the user when that branch point is reached. The applicability of each rule can be dependent on the path the user has followed up to that point. To maximize personalization opportunities, we allow a creative developer to define multiple individualized branch-point rules for the same playback point in the underlying media. To break ties when there are multiple branch-point rules that apply, we employ a longest-path matching policy that always picks the rule that most closely specifies the conditions for when the rule should be applied.
- **Branch-point aware buffer management and rate-adaptation:** To ensure seamless playback with minimum playback interruptions, while allowing the user to defer playback decisions as late as possible, careful buffer management and prefetching is required. For each branch point, the player prefetches fragments for the different candidate paths into a prefetch buffer and manages the content of the playback buffer as needed. To achieve the best possible playback experience the qualities of the prefetched fragments are adaptively selected based on estimates of the download rate and the likelihoods of the different path choices. While we leave a deeper evaluation for future work, we provide some promising validation results for one such candidate policy.

To the best of our knowledge, no prior work has combined HAS and multi-path video. We argue that merging these two ideas is natural and allows highly personalized delivery for heterogeneous clients.

The flexibility of our generalized multi-path format and the seamlessly linked media sequences is attractive when offering personalized or customizable versions of many existing services (such as news on-demand, virtual tours, etc.). However, perhaps more importantly, the simplicity of our solution opens the door for user-generated multi-path nonlinear video; creating an endless possibility of new opportunities and media forms.

While our solution and system design (Section II) applies to any HAS-based media, our proof-of-concept implementation (Section III) is implemented as an extension to Adobe's Open Source Media Framework (OSMF). Our preliminary validation results (Section IV) are encouraging and show that the example implementation can adapt prefetch qualities based on current conditions such as to avoid stall events at branch points.



(a) Nonlinear segments

(b) Multi-path video

Figure 1: Example media structure.

2. SYSTEM DESIGN

2.1 Design goals

While companies are spending much money on personalization of information and services, little attention has been given to personalizing the video sequences presented to individual users. Typically, all viewers are expected to watch the entire video sequentially. We present a novel system design that allows seamless personalized video delivery. Our solution is motivated by five primary design goals:

- **Flexibility:** The solution should allow creation of a wide variety of multi-path, nonlinear media structures.
- **Personalization:** The solution should make it easy to personalize the viewer experience based on both information about the user and previous path choices made by the user.
- **Privacy:** Personalization of the viewer experience should be possible based on both information that the user is willing to share with the server (i.e., public information) and information that the player keeps private.
- **Scalability and light-weight overhead:** The solution should avoid creating, storing, and delivering unnecessary redundant file data. For example, by avoiding any changes to underlying linear media objects and their manifest files, greater scale can be achieved as different multi-path, nonlinear videos can be created from the same stored video objects.
- **Adaptive and seamless playback:** To ensure seamless playback and effective resource usage for heterogeneous clients, the solution should include prefetching, as well as quality-aware rate adaptation.

In addition to the above primary goals, extensions are also possible which can enhance our solution. For example, to improve the user experience (e.g., as measured by the average playback quality) or the degree of personalization, the server could optimize and personalize metafiles leveraging information obtained based on previous user interactions with the service. This paper focuses on the first four goals and presents a proof-of-concept validation of the fifth.

2.2 Multi-path, nonlinear video

Before going into the details of our solution, we first define our terminology and general media structure. In contrast, to previous works (e.g., [6,15]), which typically use the term nonlinear media to refer to media in which the user can take different paths, we build our terminology around the playback patterns based on the original linear media object.

- **Fragments and segments:** With HAS, the video is downloaded in units a few seconds long that we term *fragments*. We term a sequence of fragments a *segment*. Note that some authors use the term segment for what we call a fragment.

- **Nonlinear segment:** A sequence of possibly non-contiguous fragments of video that are stitched together. For example, in Figure 1(a) a nonlinear segment (S_k) is defined which has cut out the fragments between 10 and 20, as well as between 25 and 27.

- **Multi-path video:** Any video in which users may take different pre-defined paths through the media. In most cases, such video can be described using a directional graph structure in which each edge corresponds to a (nonlinear) segment, and each vertex with two or more outgoing edges corresponds to a path choice. Figures 1(b) and 3 provide two such examples.

We allow fragments to appear in multiple nonlinear segments, and segments can be present along many different paths. This multi-path nonlinear video structure allows any arbitrary graph structure and path choices to be defined.

2.3 System overview

Our solution is client-driven. No server-side modifications are required; the server simply serves each fragment request.

As summarized in Section 1.3, our design include three novel components. Most of our design goals are achieved by incorporating a novel light-weight metafile that defines the multi-path structure and provides the necessary information about upcoming path selections that can be used by our player and its buffer management and rate adaption policies. The addition of a separate light-weight personalizable metafile helps separate the multi-path nonlinear information, potentially unique to each client, and the HAS manifest that describes the underlying media object and the playback qualities in which each fragment can be downloaded.

Before beginning playback, clients download their individual metafile and the common manifest files (potentially from different servers). During playback, our modified HAS player identifies the next branch point rule that the client will reach, manages buffers, and carefully prefetches video associated with the alternative path choices, based on the bandwidth conditions, available video encodings, and the relative likelihood of each path choice. Overall, good prefetching and rate adaption policies must weigh high video playback quality against the risk of playback interruptions.

2.4 Metafile structure

Our multi-path nonlinear metafile provides developers with maximum possible flexibility, and is designed to address the first four design goals of our system. The metafile allows the multi-path structure to be defined using two components:

- **Nonlinear segments:** At the convenience of the creator, a nonlinear segment can be specified as a list of specific fragments (or fragment sequences), stitched together from the original video, and a short-cut label that allow easy referencing.

- **Branch-point rules:** The overall path structure is defined using branch-point rules, with each rule specifying: (i) the characteristics for the client for which the rule applies, (ii) the path sequences that the client must have taken for this branch point rule to apply, (iii) the available path choices at the branch-point, (iv) their respective priorities, and (v) the corresponding explanations to be presented to the user when selecting between the path choices.

Branch rules can be used in two ways: user-driven or knowledge-driven choices. The most obvious way is to present clients with path choices, define priorities with which the initial fragments for each of those path choices should be prefetched, and allow the client to pick among these path choices. The second way is for the player to make the decision for the user based on private information about the client or other internal knowledge about the client's previous path choices, for example, at the time the branch point is reached. Note that this information may not be known at the time the metafile is downloaded and/or created.

Our approach reduces the overhead at the media server and ensure that the server does not have to maintain individual copies of the content and/or maintain state information on a per-client basis during the playback itself. The separation between metafile and manifest file allows the client to download the metafile from any Web server, and the media server must only serve requests for individual fragments.

A common metafile can be created and shared with clients requesting access to the service, or personalized metafiles can be created at the time of the client request. In both cases the clients independently request and download the parts of the original media content that matches its personalized and/or selected media paths.

2.5 Longest-path matching

Developers should be able to define branch-point rules that take into account the user's previous path choices. To allow this functionality, we allow the developers to define multiple branch-point rules, including multiple rules that have the same lead-in segment (going into a branch point).

To break ties when there are multiple branch-point rules that match a particular user's current playback point, we employ a longest-path matching policy that picks the rule that satisfies the properties of the client (part (i) of the rule) and for which the branch-point path (part (ii) of the rule) most closely matches the path taken thus far by the client.

2.6 Prefetching and rate adaptation

To allow seamless playback we carefully prefetch content for each candidate path for upcoming branch points. For this purpose, we maintain a prefetch buffer, which works in tandem with the playback buffer to provide uninterrupted playback experience. While ongoing work includes the evaluation of alternative buffer management designs, Figure 2 illustrates our current high-level design. To allow easy access to prefetched fragments associated with different paths,

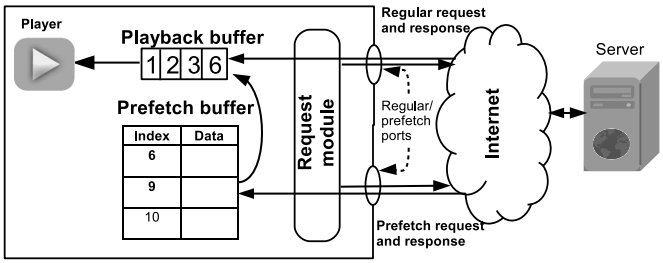


Figure 2: High-level design and buffer management.

the prefetch buffer is implemented as a dictionary with the fragment index as the key.

A request module manages the fragment downloads. It is designed to differentiate between regular fragments and *landing fragments* that are the first fragments of segments following a branch point. It is also responsible for quality-encoding selection of each requested segment.

Based on the class that a fragment belongs to, it is directed either directly to the playback buffer or is stored in the prefetch buffer. Fragments belonging to the default path are handled using the default player routines, and are immediately forwarded to the playback buffer, from which they are played in the order they are inserted into the buffer. Prefetched fragments, belonging to alternative candidate segments are typically prefetched and stored in the dictionary, from which they are moved into the playback buffer only at a time when the user selects such a non-default path. At the time of such selections, our modified player replaces the preloaded fragments for the default path with the prefetched fragments corresponding to the selected path.

Figure 4 illustrates the prefetching and buffer management process for a client that is playing a multi-path video shown in Figure 3 and is selecting the path along the red (dashed) arrows; in no case the default path. The request and download completion times are shown for each fragment. In this example scenario, we have used a policy in which the player requests the landing fragments of each path choice three fragments ahead of reaching the next branch point.

Our request module adapts the requested video quality of both the regular requests and the prefetch requests. As with other adaptive players, we keep track of the average download rates observed over previous fragments, estimating the available bandwidth, and selecting fragments qualities based on this information. While our framework allows a variety of policies, we note that a good policy should ensure that all fragments are downloaded by the time of their (potential) playback deadlines, and yet ensuring that the playback quality is of highest possible quality. For further improvements, policies can carefully weigh the priorities of different path choices; e.g., as defined by the likelihood a path is selected.

3. IMPLEMENTATION DESCRIPTION

We built an implementation of our solution using the Open Source Media Framework (OSMF) version 2.0, and its accompanying Strobe Media Playback (SMP) player.

3.1 Implementation overview

At a high level, we have added functionalities required to track, monitor, and play multi-path nonlinear video. This includes (i) changes to ensure that we can download and read metafiles, (ii) the introduction of a request module that is

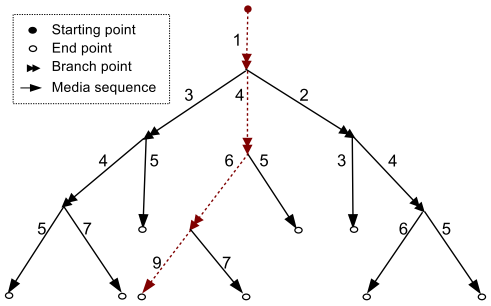


Figure 3: Example structure of a multi-path video.

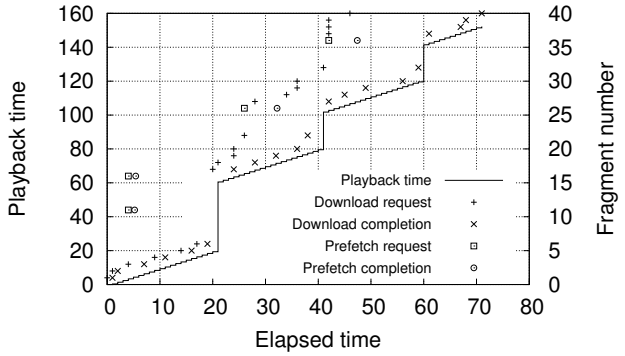


Figure 4: Timestamps when an example client request and download different segments.

responsible for prefetch requests, and (iii) the addition of a buffer manager module that is responsible for seamless playback when jumping between fragments in different parts of the original media object.

A new `downloadManager` class is responsible for downloading the multi-path nonlinear metafile and the manifest file of the original media, as well as controlling the request module. The request module incorporates both the `HttpStreamDownloader` class, which is responsible for the sequential in-order requests, as defined by the default path, and the prefetch requests. The buffer manager module works in tandem with the request module, and is responsible for controlling the fragments moved in and out of the playback buffer.

The `downloadManager` is also responsible for tracking the playback path, managing the set of branch points, and determining the next branch point to be considered.

The `downloadManager` also integrates user-triggered event actions. We use a routine in the user interface (the SMP player) as a trigger for these routines. While the user in theory should have the freedom to change the path selection up until the time of the branch point, we use a 50ms threshold before the end of the playback of the last fragment before the branch point to trigger these events.

Currently we have two versions of the SMP player. The first, more general, implementation allows for up to ten different path choices at each branch point, and path choices are entered using the keyboard. The last pressed key at the time the branch-point event is reached determines the selection. The second implementation assumes a binary tree, and use a button with a left-turn or right-turn arrow, which when pressed switches the turn of the arrow. Both implementations help the user to visualize the selected path choice, and allow changes until the branch-point is almost reached. Future work includes more user-friendly designs.

3.2 Buffer management

The interaction between our request module and buffer manager module are of critical importance in providing seamless interruption free playback. The request module feeds the playback buffer with sequential in-order fragments and the prefetch buffer with the landing fragments of the different alternative paths for the next branch point. We use a prefetch policy in which we prefetch fragments when the player is n fragments before the branch point and the `HttpStreamDownload` class is downloading fragments along the default path directly into the playback buffer.

The playback buffer behaves like a first-in-first-out (FIFO) queue, in which the fragments downloaded are read as a byte/datastream by the SMP player in the same order they were inserted. No consideration for fragments or FLV tags are present at this time which otherwise could be used to separate different parts of the media. Unfortunately, this FIFO property limits the possibility of inserting/replacing nonlinear events, as every out-of-order fragment would be queued behind already inserted in-order fragments. To allow seamless and non-interruptive playback without the user having to watch any irrelevant fragments (corresponding to non-wanted fragments along the default path, for example) which have already filled the buffer, we must therefore modify the buffer content. This functionality is implemented in our buffer management module.

At the time a non-default path is selected, the buffer management module use the `HTTPNetStream` class to invoke a seek event. When triggered, the seek routine immediately clears the playback buffer, and reinserts data from the point at which the user wants to seek to. In our case, we seek to the fragment just before the landing point. This would immediately cause the player to request the desired landing fragment. As we already have this landing fragment in our prefetch buffer, we can intercept this request and modify the request to the following segment along that path. The reason for the manipulation of requests, is to preserve some of the player's internal states and to make the player consume a fragment that was not downloaded through its normal download port. As soon as this request is placed and when the player expects some bytes, we can push the data held in the prefetch buffer into the playback buffer, which immediately can be played out. In the meantime the next fragment is downloaded and is typically downloaded by the time the landing fragment is played. All requests following this can be handled in the regular manner, without any manipulation of the player state.

By prefetching data in advance and pushing it into the playback buffer exactly when it is needed, we are able to provide un-interrupted playback even in the case of nonlinear playback events. To the user, this appears as a seamless jump from one playback point to another. Of course, there are some minor delays associated with empting the playback buffer, updating the player states, and inserting the already prefecthed fragment(s) into the player. However, during transition points between media segments these delays are minimal and are not noticeable by the user.

3.3 Rate-adaptive prefetching

Rate adaptation is at the core of any HAS-based system. Our request module utilizes the bandwidth estimations provided by the underlying OSMF player when selecting qualities, but also takes into account that both the regular requests, downloading fragments along the default path, and the prefetch requests must share the same bandwidth.

In this paper we present preliminary validation results for one basic example policy. With our adaptive policy, the request quality of the regular requests are adjusted based on the bandwidth that the prefetch requests will require. At a time when parallel downloads will be done, the qualities are selected such that all the paths are given at least the minimum quality, the quality of the default path is maximized, and all assignments are done under the constraint that their combined estimated bandwidth should not exceed the overall estimated available bandwidth. For validation purposes we also include results for a policy that does not perform any prefetching, and hence does not require any additional buffer management. Future work includes in-depth investigation of a range of policy alternatives.

4. PROOF-OF-CONCEPT VALIDATION

4.1 Experimental setup

All experiments were done in a testbed, in which we connect a client machine and a server machine over a LAN. Running `dummynet` at the client machine, we can manipulate the perceived bandwidth conditions, end-to-end delay, and packet losses. Our modified player is embedded in a Web page and runs on the client PC. The server runs Adobe Media Server version 5.0, and hosts a video of the animation movie Big Buck Bunny, encoded at 1300Kb/s, 850Kb/s, 500Kb/s and 250Kb/s. We have also instrumented the source codes to write internal player states and information into a log file, which we post process for further analysis.

4.2 Player validation

To validate the functionality of our multi-path nonlinear player, we performed a number of validation experiments. Due to limited space we can only show a small subset.

Table 1 summarizes the stall-time statistics for validation experiments in which we used a low (1Mb/s), intermediate (2Mb/s) and high (3Mb/s) bottleneck bandwidth, and the client used the test path in Figure 3, with each segment consisting of five fragments. Figure 5 presents the video quality statistics for these experiments. Average values and standard deviations are presented over ten experiments.

We can see that our adaptive policy nicely adjusts the video quality to current conditions, and is able to completely eliminate any playback interruptions due to the data not being delivered to the client in time. At this time, the only waiting times the player will endure is the time that it takes to clear the buffer and reload the prefetched content. We see that this time is typically around 0.5s, out of which 0.2-0.3s is to load the content and the rest is to update various internal states in the player. We have found that this time is not noticeable to the human eye, as long as we shift scene.

Our initial results are encouraging. However, many interesting performance tradeoffs remains to be considered, including tradeoffs between video quality and potential playback interruptions. Future work includes the development and performance evaluation of more advanced buffer management and rate adaptation policies.

5. RELATED WORK

In the past, much work has considered scalable server-side protocols for on-demand media delivery [7, 10], including

Scenario	Policy	Late data (stall events)	Branch time (seconds)
3Mb/s	No prefetching	100%	3.39 (0.94)
	Adaptive prefetch	0%	0.49 (0.10)
2Mb/s	No prefetching	100%	4.96 (1.08)
	Adaptive prefetch	0%	0.64 (0.19)
1Mb/s	No prefetching	100%	4.14 (1.10)
	Adaptive prefetch	0%	0.68 (0.17)

Table 1: Stall events and branch times.

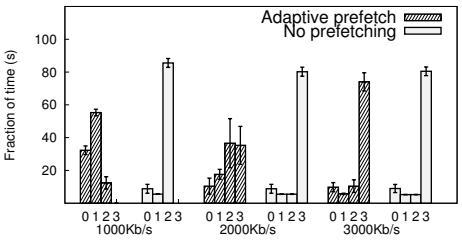


Figure 5: Playback quality.

some works that use scalable techniques to deliver multi-path (traditionally referred to as nonlinear) media [3, 6, 15]. For example, Zhao et al. [15] present lower bounds on the server bandwidth requirements for different protocols and media objects when incorporating multicast-based delivery techniques such as stream merging and periodic broadcast protocols. In general, these solutions divide the media into different segments (or pieces) that are multicast/broadcast on dedicated channels to which clients can tune in. In contrast to these works, we do not require any multicast/broadcast capability from the server.

Other related work considers different means of encoding and delivering 3D content [5], proposes a syntactic language to describe nonlinear media [14], and develops video authoring tools [12]. These works split the file into many smaller objects that can be downloaded independently. Meixner and Hoffmann [11] also present different download and caching strategies to facilitate interruption free nonlinear playback. In contrast to these works, we leverage the existing structure of HAS, which in addition to pre-defined chunking, also allows for adaptive prefetch quality based on available bandwidth conditions. This property is particularly important when prefetching video content for upcoming branch points.

Finally, Johansen et al. [8] leverage annotated video and search functionality to concatenate multiple video clips (back-to-back) that may be of interest to the users. Combining such server-side search-based functionality with our solution could potentially further enhance the user experience.

6. CONCLUSION

This paper presents a novel system design that leverages the fragment-based nature and differentiated quality levels of HAS to define and achieve seamless streaming of multi-path video. Personalization of the content can be achieved with the use of a customized metafile. Our overall solution is simple and provides the creative home user with the power to create inventive multi-path, nonlinear videos and to explore new media forms. While preliminary validation results are presented, future work includes more careful exploration and performance evaluation of more advanced buffer management, prefetching, and rate adaptive policies. The impact of caching policies may also provide interesting future avenues of research [9].

7. ACKNOWLEDGEMENTS

The authors are thankful to our shepherd Eduardo Cerqueira and the anonymous reviewers for their feedback, which helped improve the clarity of the paper. This work was supported by funding from Center for Industrial Information Technology (CENIIT) and the Swedish National Graduate School in Computer Science (CUGS) at Linköping University, the Natural Sciences and Engineering Research Council (NSERC) of Canada, and National ICT Australia (NICTA).

8. REFERENCES

[1] S. Akhshabi, A. C. Begen, and C. Dovrolis. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP. In *Proc. ACM MMSys*, San Jose, CA, Feb. 2011.

[2] A. C. Begen, T. Akgul, and M. Baugher. Watching video over the web: Part 1: Streaming protocols. *IEEE Internet Computing*, (15):54–63, 2011.

[3] N. Carlsson, A. Mahanti, Z. Li, and D. Eager. Optimized periodic broadcast of nonlinear media. *IEEE Transactions on Multimedia*, 10(5):871–884, Aug. 2008.

[4] J. Erman, A. Gerber, S. Sen, O. Spatscheck, and K. Ramakrishnan. Over the top video: the gorilla in cellular networks. In *Proc. ACM IMC*, Germany, Nov. 2011.

[5] A. Gotchev. Computer technologies for 3d video delivery for home entertainment. In *Proc. CompSysTech*, Gabrovo, Bulgaria, 2008.

[6] D. Gotz. Scalable and adaptive streaming for non-linear media. In *Proc. ACM Multimedia*, 2006.

[7] A. Hu. Video-on-demand broadcasting protocols: a comprehensive study. In *Proc. IEEE INFOCOM*, 2001.

[8] D. Johansen, P. Halvorsen, H. Johansen, H. Riiser, C. Gurrin, B. Olstad, C. Griwodz, øA Kvalnes, J. Hurley, and T. Kupka. Search-based composition, streaming and playback of video archive content. *Multimedia Tools and Applications*, 61:419–445, Nov. 2012.

[9] V. Krishnamoorthi, N. Carlsson, D. Eager, A. Mahanti, and N. Shahmehri. Helping hand or hidden hurdle: Proxy-assisted http-based adaptive streaming performance. In *Proc. IEEE MASCOTS*, Aug. 2013.

[10] A. Mahanti, D. L. Eager, M. K. Vernon, and D. Sundaram-Stukel. Scalable on-demand media streaming with packet loss recovery. In *Proc. ACM SIGCOMM*, San Diego, CA, Aug. 2001.

[11] B. Meixner and J. Hoffmann. Intelligent download and cache management for interactive non-linear video. *Multimedia Tools and Applications*, pages 1–44, Jun. 2012.

[12] B. Meixner, K. Matusik, C. Grill, and H. Kosch. Towards an easy to use authoring tool for interactive non-linear video. *Multimedia Tools and Applications*, pages 1–26, Sept. 2012.

[13] A. Rao, A. Legout, Y.-s. Lim, D. Towsley, C. Barakat, and W. Dabbous. Network characteristics of video streaming traffic. In *Proc. ACM CoNEXT*, Tokyo, Japan, Dec. 2011.

[14] A. Sobe, L. Bszrmenyi, and M. Taschwer. Video Notation (ViNo): A Formalism for Describing and Evaluating Non-sequential Multimedia Access. *International Journal on Advances in Software*, 3(1/2):19–30, Sept. 2010.

[15] Y. Zhao, D. Eager, and M. Vernon. Scalable on-demand streaming of nonlinear media. *IEEE/ACM Transactions on Networking*, 15(5):1149–1162, Oct. 2007.