The Media State Vector[*]

A unifying concept for Multi-device Media Navigation

Ingar M. Arntzen
Northern Research Institute
Tromsø, Norway
ingar.arntzen@norut.no

Njål T. Borch
Northern Research Institute
Tromsø, Norway
njaal.borch@norut.no

Christopher P. Needham
BBC R&D
London, UK
chris.needham@bbc.co.uk

ABSTRACT

This paper presents the concept of the Media State Vector (MSV), a representation of uni-dimensional motion in real time, intended as a general basis for synchronization of distributed media presentations. The MSV concept is motivated by the idea that motion can be decoupled from media presentations, and that media presentations can be modeled as client-synthesized functions of motion. Implementation of the MSV concept for the Web allows us to construct navigable, synchronized, multi-device, multimedia presentations, spanning computers across the Internet. In particular, media presentations may be hosted by regular Web browsers on a range of devices, including smart phones, pads, laptops and smart TVs. Our proof of concept implementation bases its accuracy on primitive, centralized, ad-hoc, application-level clock synchronization. Still, in our experiments 80% of clients were able to synchronize across Europe, with a maximum error less than 120 ms. Inter-client synchronization error of about 33 ms is demonstrated between three screens in London (UK), synchronized via a server in Tromsø (Norway). These results, along with the lightweight, scalable, generic and minimalistic nature of MSV synchronization, lead us to propose the MSV concept as a foundation for cloud-based media synchronization.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; H.5.1 [**Information Interfaces and Presentation**]: Multimedia Information Systems

General Terms

Design, Theory

[*]This work was funded in part by the Seventh Framework Programme (FP7/2007-2013) under grant agreement No. 21617 (P2P-Next), and Interreg North (IV/A) under norwegian grant no. IR 12/11.

Keywords

Mobile, cloud computing, media synchronization, Web

1. INTRODUCTION

We perceive media presentations as combinations of *content* and *navigation*. For instance, consider any media event, be it a major sports event, a riot, or simply a gathering of friends. The media output may include a variety of content types, ranging from images, Twitter messages and amateur videos to TV news reports, radio interviews and newspaper articles. More elusive perhaps is the pervasiveness of media navigation. End-users *play*, *pause* and *rewind* videos, and they navigate image galleries and news feeds in stepwise fashion, forwards and backwards. Live TV broadcasts may include short segments of instant replay, possibly in slow-motion. A time-lapse is the opposite, where media content is navigated at high speed. Furthermore, time-shifting can be understood as media navigation, whether performed by the end-user on a DVR (Digital Video Recorder), or by the TV provider as the delaying of a live broadcast.

Media navigation is thus an integral part of the media landscape, yet implementations are often custom and tightly bound to media types and visual presentation. We propose a novel, unifying concept for media navigation, the *Media State Vector (MSV)*. The MSV is an encapsulation of uni-dimensional motion in real time, serving as a generic representation of media navigation. The central motivation is the decoupling of media navigation from content and presentation. Decoupling enables shared navigation for multiple presentation components and multiple content sources. Furthermore, decoupling and encapsulation allows media navigation to be shared across a network. With the emergence of cloud services for media presentations, the latter is particularly relevant. While media content is steadily migrated into cloud services, media navigation typically remains local. We conjecture that media navigation too can be hosted by cloud services using MSVs as synchronization points for clients across the Internet.

A new perspective on media presentations is attainable once content and navigation are both hosted by cloud services. Now media presentations conceptually execute, not at the clients, but *in the cloud.* Access to the media presentation is mediated through views hosted by client devices. This change in perspective has profound implications. By enabling multiple end-users to simultaneously interface with a single media presentation, each through their own view, collaborative viewing and social interaction becomes integral to media presentations. Or, by splitting content and presentation between multiple views, utilizing the strengths of different devices, a solution is provided for secondary screen scenarios.

Accurate synchronization is required to support this abstraction. If a media presentation is requested to play in one view, all views must play in synchrony. MSV synchronization does this. It is a general, lightweight and scalable approach that can be implemented in any network. However, there is a often a trade-off between accuracy and scope. MSV synchronization implemented for a LAN environment may be highly accurate, but limited to devices on the same network. In this paper we focus on synchronization of media navigation across the Internet. We argue that the latency and accuracy associated with MSV synchronization is acceptable for many media applications, given the significant benefits offered by cloud-based media navigation.

We have implemented MSV synchronization for the Web. This allows us to quantify the accuracy of synchronized, cloud-based media navigation. The availability of synchronized clocks is not assumed. In our implementation, synchronization accuracy depends on primitive, application-level estimation of clock skew. This implies that our results are conservative, and that accuracy can be improved by the use of better estimation techniques and access to low-level network and timing primitives. Equally important, it implies that any networked device is allowed to participate in synchronized, cloud-based media navigation. In particular, standards compliant Web browsers can readily host synchronized presentation views without requiring additional plugins or modification.

2. PROBLEM DEFINITION

The main objective for this thesis is to propose a general mechanism for Web-based, multi-device media navigation. In the interest of generality, we deliberately avoid a discussion of particular media applications. Instead, requirements are derived from the abstract idea of a media presentation implementing *the cloud abstraction*. By this we mean 1) a media presentation that can be perceived as being a single entity residing in the cloud, and 2) that access to media presentations is mediated through views hosted by client devices.

1. *Synchronous Navigation.* All participating views shall navigate the media presentation in best-effort synchrony, at all times. Navigational changes requested in one view shall affect all views as quickly as possible.

2. *Symmetric Navigation.* All participating views shall be equally able to control common navigation.

3. *General Navigation.* A wide range of navigational primitives shall be supported, including primitives implemented by traditional categories such as e-books, audio, video, animations and slideshows.

4. *Device Heterogeneity.* Any networked device shall be able to host a view into a synchronized media presentation, including smart phones, pads and smart TVs.

5. *Content Heterogeneity.* Presentation views shall not be limited to a single media type or a single transport mechanism. Multiple media types (e.g., text, images and videos) may be part of a single view presentation.

6. *View Heterogeneity.* Devices shall be able to present different views related to the same media presentation. This way, devices with interactive ability may host views for navigation, selection and text input, whereas TVs may be used primarily to host visual components such as video, pictures, or maps.

7. *View Autonomy.* Participating views shall not depend on other participating views.

These are overall requirements. In addition there are certain requirements that relate to specific *instances* of media presentations. By, *instance* we mean the following: If two views are participating in synchronized navigation they may be perceived as connected to a single instance of a media presentation. In contrast, if two views are hosting the same content and presentation, but navigation is independent, then they are connected to distinct instances.

8. *Simple Instance Management.* It shall be easy to create (or delete) an instance of a media presentation.

9. *Dynamic Instance Membership.* All views shall be able to join or leave the media presentation at any time, without affecting other views or the presentation instance in any way. Joining views shall be able to quickly synchronize with existing views.

10. *Instance View Independence.* An instance of a media presentation shall be independent of client views. This implies that a running media presentation shall continue to run, even though no views exist.

11. *Instance Autonomy.* Multiple instances of the same media presentation shall be able to coexist, yet be independent of each other. This way, multiple groups of views may navigate the same media presentation simultaneously. Groups maintain internal synchrony, but in-group navigation shall not affect other groups.

12. *Instance Identification.* A particular instance shall be easily identifiable so that invitations to join can be shared and distributed.

These are all functional requirements. In order for the mechanism to be appropriate as basis for a cloud-based service there are also non-functional requirements. Multi-device media navigation on the Web must be highly scalable, support reliable implementation, and preferably be open.

A number of multi-device media technologies address some, but not all of the above requirements. For example, broadcast technologies such as TV, desktop-sharing, web-cast and video conferencing systems support instance view independence and view autonomy, but often lack ability for symmetric navigation and view heterogeneity. Secondary screen applications target view heterogeneity and device heterogeneity, but typically do not support symmetric navigation or view autonomy. Google Plus Hangout[1] and SynchTube[2] support synchronized navigation of YouTube videos. Both solutions are proprietary, restricted to streamed video content, and the scalability is limited to small groups.

[1]https://plus.google.com
[2]http://www.synchtube.com

3. APPROACH

Our approach is defined by a six step argument.

Arg1. Motion is common to all media presentations

All media presentations relate to some notion of directionality or progress. For example, a regular slideshow defines its progress according to a discrete axis of integers, or slide numbers. Similarly, a video is related to an axis of real numbers, usually denoting offset in seconds. A small set of operations define ways in which progress may be requested to change. Slideshow operations such as *next, prev* and *goto* cause discrete jumps between slides. Video operations such as *play, pause* and *skip* regulate progress and the rate at which progress changes (i.e., velocity). We argue that the notions of progress and navigation that are evident here can be described under a more general concept of *motion*.

Arg2. Motion is separable from media presentations

Every application that executes media presentations needs to maintain an internal representation of motion. Very often this representation is integral to both media type and visual presentation. In contrast, we argue that motions can be cleanly decoupled from media presentations, to which they are both cause and effect. Decoupling of motion from content and presentation allows us to think of motion as an object in its own right. Furthermore, it allows a single motion to be shared between multiple content sources, content types and presentation components.

Arg3. Media presentations are functions of motion and content.

We perceive media presentations as functions of motion and content. Media presentation F is at all times a function of motion m and content c: $F(m, c)$. Here, we define media content as a set of media objects, each related in some way to an abstract axis of progress. At any time, motion along this axis defines which media objects, and what parts of media objects must be presented. Thus, F maps motion and content to presentation. More complex media presentations may be functions of multiple motions and multiple content sets, e.g., $G(m_1, m_2, \ldots, m_n, c_1, c_2, \ldots, c_m)$.

Arg4. Synchronize motion, not content

Our approach to synchronization of media presentations is defined in terms of synchronization of motions. Two media presentations F and F' are said to be synchronized if a motion from F is synchronized with a motion from F'. With this definition two media presentations can be synchronized, although their visual (or audio) appearances have nothing in common. This differs from traditional approaches, where synchronization is often defined in terms of similarity of presentation. However, note that motion-based synchronization may also be used to achieve similarity. By synchronizing *all* motions of two *identical*[3] media presentations F and F', identical presentation is approximated.

Arg5. Motion is an object on the Web

Synchronization between different devices requires synchronization of motion across a common network. We facilitate synchronization of motion by making motion an object on the Web, i.e., available on the Internet using the HTTP protocol. This way all connected devices, including Web browsers, may participate in synchronized media presentations simply by being clients to the same instance of motion. In short, we are migrating instances of motion from the private world of client applications into the public world of the Web.

Arg6. Client-side synthesis of media presentations

We advocate a world where media presentations are synthesized in real time by client devices, in response to motion. There are several reasons why we think this is feasible. First, modern client devices increasingly feature high computational power and high bandwidth Internet connections. Second, client-side synthesis is a foundational idea of the Web. Web browsers synthesize documents from collections of resources. Similarly, client devices may weave together media presentations in real time, from multiple media types and content sources. Third, client-side synthesis implies that media presentations can integrate with current and future services of the Web, while executing.

In summary, this approach to organizing, executing and synchronizing media presentations promises to be liberating in several ways. It releases navigable media presentations from the strict limitation of being single-device. It also does away with the limitations of being single content source, single transport mechanism, single media type, single visualization or single user.

The scope of this paper is limited to the challenge of synchronizing motions across a network. We do not discuss specific applications of motion synchronization. Furthermore, challenges concerning the implementation of media presentations as functions of motion and content are *not* covered. This includes common issues such as media specification, timely content delivery, and synchronization of content relative to motion. We refer to these topics only briefly in related work, Section 8.

4. THE MEDIA STATE VECTOR

The *Media State Vector (MSV)*[4] is the central concept underlying our implementation of motion synchronization. The MSV is an object that describes uni-dimensional motion in real time. It is based on a general and well known model for describing motion: the classical equations of uniformly accelerated linear motion[5]. This is a *deterministic* model, implying that any *movement* can be described in terms of *initial conditions* and the *time* that has passed since these conditions were set. The MSV describes *motion* as a time sequence of distinct, deterministic *movements*. The MSV defines two operations, *query* and *update*. Query returns a snapshot derived from the current *movement*. Update replaces the current *movement* with a new one.

4.1 Internal State

The internal state of an MSV is a vector with four real numbers $[p, v, a, t]$ representing initial position (p), velocity

[3]Media presentations F and F' are identical iff $m = m' \Rightarrow F(m, c) = F'(m', c)$ $\forall$ values of m. Here we assume c to be shared and static.

[4]MSV demo at http://goo.gl/QuHzN

[5]Uniformly accelerated linear motion was chosen because it is well known and maps well to common navigation in media presentations. The crucial point is that the model is deterministic.

(v), acceleration (a) and time (t). Position fixes an abstract point to a one dimensional axis at time t. Time t is expressed in seconds. Velocity denotes the rate at which position is changing at time t, expressed in position-units per second. Similarly, acceleration denotes the rate at which velocity is changing at time t, expressed in position-units per second squared. The MSV concept is named after this internal state vector.

> *internal state:* $[p_i, v_i, a_i, t_i]$, where t_i is initial time of current *movement*.

4.2 Query

The MSV may be queried at any time for a snapshot derived from its current *movement*. The query operation is atomic.

> *query():* returns $[p(t), v(t), a(t), t]$, where t is time of processing the query operation, and

$$p(t) = p_i + v_i(t - t_i) + \frac{1}{2}a_i(t - t_i)^2 \quad (1)$$

$$v(t) = v_i + a_i(t - t_i) \quad (2)$$

$$a(t) = a_i \quad (3)$$

For instance, if the query operation returns $[1.2, 2.0, 0.0, t]$ at time t, then a new query a second later should return $[3.2, 2.0, 0.0, t + 1]$. It may be helpful to think of the MSV as representing a single floating point variable that changes dynamically in real time. This variable corresponds to MSV position p. In this perspective, velocity v and acceleration a are just supplementary information, describing exactly how position p changes in time. MSV position p has no prescribed unit. Instead, applications define its meaning in application specific terms, e.g., discrete slide numbers or continuous media time offset.

4.3 Update

The MSV update operation allows the current *movement* to be replaced by the next. This is achieved simply by overwriting the internal state of the MSV. The triplet (p, v, a) describes the next *movement* and are given as parameters to the update operation. The update operation is atomic, thereby ensuring strict ordering of updates.

> *update(p,v,a):* sets $[p, v, a, t]$ as new internal state, where t is time of processing the update operation. The return value is the new internal state.

The parameters for the update operation are optional, i.e., *null* values may be supplied. This provides a simple mechanism for tying *movements* together. The idea is to allow one aspect of the *movement* to be updated while *preserving* the others. For instance, $update(null, v, null)$ means update velocity while preserving position and acceleration. Note that the values we want to preserve are not from the internal state, but rather from a snapshot of the MSV, taken at exactly the time when the update is processed.

The update operation is quite expressive. For example, $update(null, 1.0, 0.0)$ and $update(null, 0.0, 0.0)$ implements *play* and *pause* operations typical for video content. By using the query operation, relative updates can be constructed. For instance, $update(floor(query()[0] + 1.0), 0.0, 0.0)$ implements the discrete *next* operation typical for slideshow and list navigation. $update(10.0, null, null)$ skips to a new position, but preserves velocity and acceleration. $update(10.0, -1.0, 1.0)$ skips to a position and starts a backwards motion with acceleration in the positive direction. Note also that clocks are a special case of the MSV, where velocity is constant. Given an epoch value in seconds (past, present or future), $update(epoch, 1.0, 0.0)$ creates the appropriate clock.

Acceleration may not be required by, or even applicable to, all types of media presentations. However, if applicable, it may offer smooth transitions between different playback velocities. This is particularly relevant for media presentations that do *not* include continuous media types (i.e. audio or video). For example, a media presentation that visualizes time series of scientific data might contain long segments of static data, or few data points. Acceleration and deceleration may then be used to smoothly navigate these segments, without losing the sense of time. Similarly, a visualization of evolutionary history may start out with bacteria at high velocity, and decelerate as the narrative closes in on the present. Indeed, in this case a motion with non-constant acceleration might be preferred. If so, such motions may be implemented by means of step-wise approximation.

4.4 Update Event

The MSV additionally supports one event (callback) associated with the update operation. This allows objects that interact with the MSV to detect updates applied by other objects, asynchronously. The update event includes a copy of the new internal state of the MSV (i.e., the result of the update operation). There is no requirement on the MSV to process an update request immediately. Interacting objects must wait for the update event (or update response) to acknowledge the effects of requested updates.

> *update event:* includes $[p, v, a, t]$, the new internal state of the MSV.

4.5 Range Restrictions

The ability to enforce range restrictions on position, velocity or acceleration is optional, but is often a helpful feature. For instance, a range on position $[0.0, 123.0]$ would be appropriate for a media presentation with a limited duration of 123.0 seconds. In general, as soon as a *movement* of the MSV violates one of its range restrictions, the *movement* needs to be replaced by a legal one. The MSV enforces its own range restrictions by protecting itself from illegal updates and by detecting range violations as time passes. The deterministic model allows the MSV to precisely predict future range violations. By means of a timeout mechanism it can then schedule the appropriate update operation on itself. In the above example $update(123.0, 0.0, 0.0)$ will be executed at the exact time playback position has reached the end. From the perspective of an interacting object, this kind of update is indistinguishable from an update requested by another object.

5. MSV SYNCHRONIZATION

Cross-network motion sharing allows distributed media components to be perceived as parts of the same media presentation. In our approach motion sharing is facilitated by making MSVs available on the Web. However, MSV synchronization is a conceptual contribution and can be implemented across any network. This section presents MSV synchronization in general terms, without referencing particular network assumptions, implementation details or application requirements.

In the following an MSV instance hosted by a server is denoted as a *server MSV*. Similarly, an MSV instance hosted by a client device is denoted as a *client MSV*. *MSV synchronization* requires a client MSV to continuously mirror the motion of a server MSV. If multiple clients mirror the same server MSV, synchronization *between* clients is achieved by implication. This is illustrated by Fig. 1. All update requests applied to client MSVs are forwarded to the server MSV. After the update has been processed, update events are transmitted to all clients. A single online MSV on the Web can be the source of motion for multiple client devices and/or multiple media presentations.

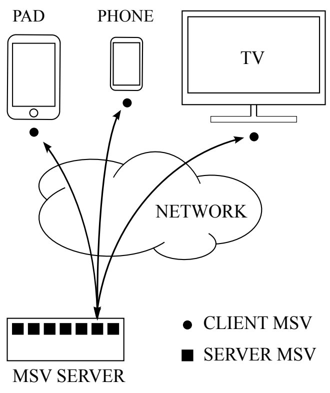

Figure 1: Three devices synchronizing their client MSVs with a single server MSV, across a network.

MSV synchronization involves two distinct challenges. First, update operations processed by the server MSV must be communicated to and processed by all client MSVs as quickly as possible. We call this *update synchronization*. Second, after processing update events, client MSVs need to maintain motion synchrony until interrupted by the next update, possibly indefinitely. We call this *regular synchronization*.

5.1 Update Synchronization

Update synchronization depends primarily on quick notification of client devices. An effective and dependable push mechanism is required. Such a mechanism allows update events to be transported from the server to clients without unnecessary delay, thus avoiding extensive cross-network polling by clients. The diagram A at the top of Fig. 2 illustrates this scenario. It portrays the real time control flow of one server and two clients (horizontal lines). At time t_1 an update is processed by the server and transmitted to its clients. At time t_2 and t_3 the update events are received by the two clients.

Transport delay *trans* is defined to be the real time interval between an update being sent by the server to it being processed by the client. Diagram A illustrates different transport delays for client 1 and 2, i.e., $trans_1 = t_3 - t_1$ and $trans_2 = t_2 - t_1$. Such variation in transport delay is a source of asynchrony for update synchronization. So, the accuracy of MSV update synchronization depends on the variation in *trans*.

It is possible to improve the accuracy of update synchronization by introducing extra delay in the client, in order to mask variation in transport delay. We discuss this option further at the end of Section 5.2.

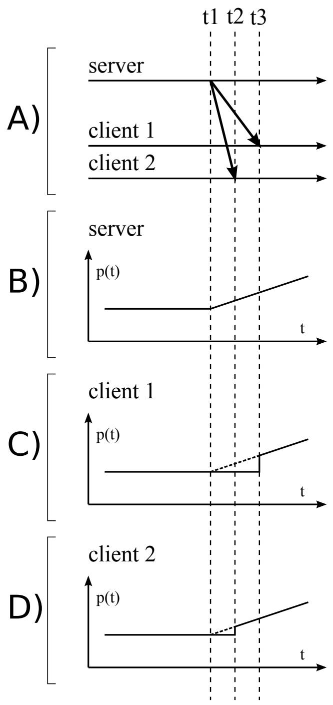

Figure 2: MSV synchronization with one server and two clients.

5.2 Regular Synchronization

Regular synchronization involves maintaining synchrony in the indefinite time interval between updates. Due to the deterministic nature of the MSV, clients require only the last update event in order to monitor and predict MSV behavior locally. The resolution of the local MSV is determined by the resolution of the local system clock. Thus clients can support a local representation of a server MSV, with high time resolution, without penalizing the network.

Still, high resolution is not enough. Ideally motions should play out simultaneously at server and client. In order to approximate this, differences between server and client clocks are taken into account. The idea is to synchronize MSVs simply by correcting for clock differences in the time element of the update event. So, $[p, v, a, t_{server}]$ must be transformed to $[p, v, a, t_{client}]$ by the client MSV. Here t_{server} and t_{client} are values of server and client clocks at the same instant in real time. To achieve this transformation we depend on clock synchronization. The client has access to a software clock $C(t)$ synchronized with the server clock. $C(t)$ takes input t from the client clock and outputs the corresponding value of the server clock. Furthermore, $E(t)$ is introduced to estimate the fixed value t_{client}. This way the internal state of the client MSV becomes $[p, v, a, E(t)]$. Estimator $E(t)$ is a function of client time t.

$$E(t) = t_{server} + t - C(t) \tag{4}$$

To verify the correctness of equation 4, consider a query processed by the client MSV. In equations 1 and 2 (see Section 4.2) the time delta $(t - t_i)$ is computed, where t refers to processing time and t_i refers to the time element of the MSV internal state. Replacing t_i with $E(t)$ yields time delta

$(C(t) - t_{server})$. If $C(t)$ is a perfect estimator of the server clock, then at any point in real time the time delta used by the client MSV is exactly equal to that used by the server MSV. Thus, the accuracy of MSV regular synchronization is determined by the accuracy of the software clock $C(t)$.

Using a client-side software clock $C(t)$ additionally implies that MSV synchronization can handle clock drift, provided $C(t)$ can. MSV dependence on $C(t)$ ultimately means that the progression of the client MSV is determined by the system clock of the server, not the system clock of the client. This is appropriate because synchronized motions should ideally play out simultaneously at multiple devices, indifferent to relative differences in clock frequency.

Fig. 2 includes three diagrams *B, C,* and *D* illustrating synchronization of motion. The horizontal axis represents increasing time, and the vertical axis represents the *position* element of the MSV, as a function of time. The time axis of each diagram is aligned so that motions can be compared. Vertical lines represent the same instant in real time.

The first diagram, *B*, illustrates the behavior of the server MSV. Initially the position is constant in time. At time t_1 an update is applied and the motion continues with non-zero velocity.

Diagrams *C* and *D* illustrate how this motion is reported by two client MSVs, *client1* and *client2*. Compared to the server MSV, both clients report *incorrect* values for a small interval $trans_i$ (i.e., the time it takes to propagate an update event from the server to a client). This is the incorrect continuation of the movement that was just interrupted at the server. However, as soon as the event updates are received, the client MSVs again report correct motion. These diagrams depict an ideal situation where $C(t)$ is perfectly estimated. Imperfect estimation will shift the client motion to the left or to the right in this diagram.

The fact that MSV synchronization reports incorrect motion during a short interval might be a problem for some applications. If so, the problem can be avoided by delaying motions by a fixed *delta* on all clients equally. As long as *delta* is set larger than the sum of the maximum *trans* and the maximum error in $C(t)$, update events will be received by clients *before* they are due. This approach preserves the integrity of the motion, but increases update latency. Long update latency may be confusing for the end-user, unless it is properly communicated in the user interface. In addition, delaying motions will mask variation in *trans* between clients, as noted in Section 5.1. This strategy implies that the accuracy of update synchronization becomes equal to the accuracy of regular synchronization. Motion delay can be implemented in MSV clients by replacing software clock $C(t)$ with a modified version $C'(t) = C(t) - delta$. In effect, a positive *delta* shifts the server clock back compared to the client clock. Update events must also be delayed and applied according to this modified server clock $C'(t)$. *delta* is considered a property of the server MSV, available for all MSV clients.

MSV synchronization errors, i.e., errors in $C(t)$, manifest as errors in MSV position and velocity. However, the size of these errors will depend on the specific motion. For instance, if acceleration is non-zero, the error in $C(t)$ will be squared as position is computed. Similarly, there is no error in position if velocity and acceleration are both zero. Thus, *pause* operations will always cause synchronized MSVs to agree on position, even though clock synchronization might be inaccurate.

Note also that synchronization errors do not threaten the consistency of MSV synchronization. Update requests are serialized by the server, according to the server timeline. This ensures agreement on ordering and timing among clients, independent of inter-client synchronization errors. So, if one client struggles with a poor estimate for $C(t)$, other clients are not penalized. This is in accordance with requirement 7 of Section 2; *View Autonomy.*

On the other hand, synchronization errors do affect the integrity of the motion reported by clients during update synchronization. Especially, for clients that run ahead of the server (in real time), the synchronization error adds to the interval of incorrect motion. The integrity of a motion may also be violated if $C(t)$ experiences sudden *jumps.* There are several possible strategies for dealing with this. Cristian [1] outlines an implementation of a monotonically increasing $C(t)$ where jumps are implemented gradually. Another strategy is to integrate jumps only when new update events are applied to the client MSV. It is also possible to enforce a short bootstrapping period so that $C(t)$ becomes reasonably stable before the MSV is used. The best strategy is likely to be application dependent.

Finally, since $C(t)$ can not in general be known a priori, an ad-hoc estimate must be created as clients initiate MSV synchronization. In Section 6 we outline how Web browsers may implement this at application level and the accuracy that can be expected for this approach in the real world.

5.3 Meeting the Requirements

In Section 2 we asserted a number of requirements for synchronized media navigation supporting navigable, multi-device media presentations. Here we briefly argue that MSV synchronization meets these requirements. *Synchronous Navigation (1)* and *Symmetric Navigation (2)* are ensured by the combination of update synchronization and regular synchronization. The generality and expressiveness of the MSV concept supports *General Navigation (3).* The availability of MSVs on the Web, and the simplicity of the synchronization protocol imply that any networked device may participate in MSV synchronization, thus *Device Heterogeneity (4)* is achieved. MSV synchronization is also content agnostic, thereby facilitating *Content Heterogeneity (5)* and *View Heterogeneity (6). View Autonomy (7)* is a consequence of the client-server architecture underlying MSV synchronization. In addition, all the requirements concerning presentation instances relate to MSV instances hosted by MSV servers. Trivial operations to create and delete MSV instances ensure *Simple Instance Management (8). Dynamic Instance Membership (9)* is achieved by letting devices freely connect and disconnect from MSV instances. *Instance View Independence (10)* is ensured since MSV instances exist independent of client views. *Instance Autonomy (11)* is achieved by creating multiple MSV instances for a single media presentation. URIs to MSV instances are used for *Instance Identification (12).* Finally, the light-weight nature of MSV synchronization suggest that it can be the basis for highly scalable synchronization services.

6. IMPLEMENTATION

We have implemented the MSV concept for the Web, in the interest of wide applicability, and with the intent of demonstrating the feasibility of Web-based media navigation. The implementation targets the real world scenario; a range of devices using different network carriers and a variety of Web browsers. This scenario informs our design choices.

First, we require that our implementation of MSV synchronization works in standards compliant Web browsers, without requiring any plugins or other modifications. Second, we do not assume the availability of synchronized clocks, nor do we assume access to low level network or timing mechanisms. This means that cross Internet clock synchronization (See Section 5.2) will have to be implemented at the application level, using the network and timing mechanisms available in regular Web browsers. Third, to make sure our results are conservative we are satisfied with a primitive implementation of clock synchronization. Note however that future developments may affect these limitations. In particular, availability of NTP [2] synchronized clocks in Web-browsers would imply that MSV synchronization accuracy would be determined by the accuracy of NTP. This is an attractive avenue for further research.

We have made a reference implementation of an MSV Server in Python. In addition we have implemented a JavaScript library allowing regular Web browsers to participate in multi-device media presentations. Both the MSV Server implementation and the JavaScript library are open source[6]. This section briefly presents the MSV Server, the JavaScript library and our primitive approach to clock synchronization.

6.1 MSV Server

The MSV Server[7] is an HTTP server capable of hosting MSV instances. Each MSV instance is identified by a unique URI. The MSV Server supports *query* and *update* operations targeted at specific MSV instances. These requests are constructed by supplying query parameters to the *MSV_URI*. In addition the MSV Server supports on-demand creation and deletion of MSV instances. The MSV Server responses are JSON data, each less than 500 bytes. Table 1 defines a RESTful API for the MSV Server. A GET request to MSV_URI without parameters returns a HTML representation of the MSV.

Table 1: RESTful MSV Server API

Operation	Method	Path	Parameters
query	GET	MSV_URI	cmd=query
longpoll	GET	MSV_URI	cmd=longpoll
update	POST	MSV_URI	p=0.0&v=1.0&a=0.0
create	POST	SERVER_URI	range=[0.0,123.0]
delete	DELETE	MSV_URI	

An efficient and reliable push mechanism is a crucial aspect of the MSV Server implementation. We have implemented this by means of long polling[8] (without the use of persistent connections). In order to be notified of MSV updates each client needs to keep one connection open to the server. Then, as an update event occurs at a given MSV, the server sends the update event to appropriate clients via open connections. After this the connections are closed and the clients have to reconnect to receive further event updates. Web browsers (or network intermediaries) may cause unused connections to be terminated after a while. To avoid this the MSV Server forces any client to renew its long poll connection when it has been unused for a certain period, say 30 seconds. While pipelining over persistent connections or Web Sockets[9] are alternative solutions, long polling was chosen in the interest of simplicity and wide applicability.

The MSV Server maintains one open connection per client. An *MSVSet* abstraction is implemented to let a single client synchronize with multiple MSVs without consuming more than a single long poll connection. For our reference implementation the number of open connections is a limiting factor with respect to scalability. However, existing cloud services indicate that it is possible to scale services considerably in this regard.

6.2 JavaScript Library

The JavaScript library hides the complexities of MSV synchronization and makes production of synchronized media presentations possible for any Web developer, requiring only a basic understanding of the MSV concept. Web developers need only relate to a local MSV proxy that continuously mirrors the motion of a server MSV. The API of the MSV proxy is very similar to the general MSV API suggested in Section 4.

The library additionally allows creation of purely *local* MSVs, i.e., *not* dependent on any server MSV. Because local MSVs and MSV proxies implement the same interface, they are interchangeable as sources of motion. The ability to switch between local and remote motions provides an elegant mechanism for dealing with temporary connection failures as well as periods of *offline* media consumption. It also enables end-users to deliberately switch between *private* and *shared* scope for media navigation while the presentation is running. Note also that design and development of presentation views is exactly the same challenge, whether the view is intended for single-screen or multi-screen media presentation.

6.3 Clock Synchronization

The current implementation of MSV synchronization is based on a simplistic implementation of a software clock $C(t)$. The transformation between server and client clock is achieved by maintaining an estimate for the relative clock *skew*.

$$C(t) = t + skew(t) \tag{5}$$

A positive *skew* means that the server clock runs ahead of the client clock. The estimate of *skew* may change in time, and is re-evaluated at least every 30 seconds.

Estimates of *trans* and *skew* are generally not known a priori. Therefore clients must estimate these parameters every time MSV synchronization is initiated, and maintain them for as long as the session continues. Measurements are collected from all exchanges between client and server. In our implementation this includes queries, updates and long poll requests. Estimation does not introduce extra network traffic. However, if needed, clients may issue redundant queries in order to expedite accurate estimation.

Fig. 3 illustrates how timestamp measurements are collected. The client sends a request at *CS*. The server receives the request at *SR*. The server sends the response at *SS*, and the client receives the response at *CR*. The server response includes *SR* and *SS* so that the client can collect all four

[6]Not publicly available yet.
[7]MSV Server demo at http://goo.gl/545e2
[8]http://tools.ietf.org/html/rfc6202
[9]http://tools.ietf.org/html/rfc6455

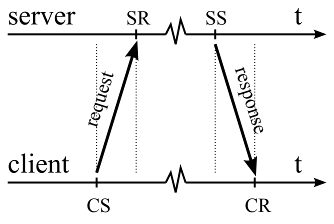

Figure 3: Client-server interaction in real time. Response may be synchronous or asynchronous.

timestamps from each exchange. Using these measurements *trans* and *skew* may be calculated as follows [2]. These formulas assume transport delays to be symmetric.

$$trans = \frac{(CR - CS) - (SS - SR)}{2} \tag{6}$$

$$skew = \frac{(SS + SR) - (CR + CS)}{2} \tag{7}$$

Measurements are obtained from a full HTTP request-response exchange[10]. We do not assume these results to be a precise characterization of the properties of the underlying network. After all, Web browsers are usually not optimized for time sensitive tasks. There is no mechanism for prioritization of time-critical communication. Multiple unrelated tasks may be running, potentially influencing time measurements. Also, the system clock available within the JavaScript environment rarely supports resolution below milliseconds.

So, the assumption of delay symmetry is very crude and is not valid for a large fraction of client-server exchanges. Our solution is inspired by Cristian's classical algorithm for probabilistic clock synchronization [1]. The assumption of delay symmetry is more likely to be sound if we consider only the client-server exchanges with the fastest round trip times. Our estimate of *skew* is continually updated to reflect the minimum *trans* measurements recorded so far. As the number of client-server exchanges grows the *skew* estimate is likely to become more accurate and stable.

7. EVALUATION

The evaluation quantifies the accuracy that can be expected from our implementation of MSV synchronization in real world scenarios. Measurements of network latency are collected by non-optimized Web browsers hosted by a diverse set of devices, on different networks across Europe. However, before measurements are presented we briefly discuss application requirements with respect to synchronization accuracy.

7.1 Application Requirements

Requirements for synchronization accuracy depend on the nature of applications, as well as the distribution of participating views and devices. A detailed discussion of MSV applications is outside the scope of this paper. Still, we argue that MSV synchronization has a wide range of applications, with synchronization requirements varying from very strict to very loose.

Scenarios that require strict synchronization have certain characteristics in common. 1) Participating devices are physically close to each other so that end-users can observe multiple devices simultaneously. 2) The media must be time-sensitive in a way that matches the sensibilities of human perception. For instance, lip-sync accuracy between audio and video is a classic example of such a scenario. ATSC[11] recommends that audio is ahead of video by maximum 15 ms, and lags behind video by maximum 45 ms. A worst case scenario might require lip-sync between a smart phone and a TV, with the two devices having network connections of poor, or very different quality. Our primitive implementation of MSV synchronization cannot guarantee the accuracy required by this scenario. However, it can get close, and the final fine tuning may be done with the help of complementary synchronization mechanisms, e.g., techniques based on digital watermarks [3]. Alternatively, end-users could be allowed to adjust this manually through an appropriate presentation view. In any case, MSV synchronization may serve as part of the solution.

Equally important, there is a considerable set of applications where the accuracy requirements are much more relaxed. For instance, the secondary screen scenario might involve enriching a TV broadcast with synchronized media on a handheld device. As long as the role of the secondary device is to play background music, present illustrative pictures, maps, graphics, tables or subtitles, the timing requirements may be in the range 100–1000 ms[12], or even more. Furthermore, if participating devices are *not* simultaneously observable by any end-user, the accuracy requirements are relaxed simply because it is more difficult to detect synchronization errors. Remote learning and collaborative viewing may be application domains where this is the case.

7.2 Experiments and Results

Network measurements have been collected from a set of MSV clients hosted by Web browsers on a diverse set of devices. The tests were run by members of the P2P-Next project, counting 76 independent experiments over two days, all targeting the same MSV server in Tromsø. Each experiment collected timestamp samples (i.e., *CS, SR, SS, CR*) from about 24 client-server exchanges taken during a 30-second session. Project members were encouraged to perform multiple experiments with different browsers on different devices. Google Analytics revealed the following statistics for our experiments. The geographical origin of clients were mostly European. A majority of experiments were performed by Norwegian, British and Dutch clients. A few experiments were also contributed from China. Experiments were conducted from 11 countries in total. Mobile devices were used in at least 17 of the experiments, including a variety of Android and iOS devices. The participating Web browsers were identified as Chrome (40%), Firefox (16%), Android Browser (14%), Safari (14%), Opera (4%), Mozilla (1%) and Netscape (1%). Internet Explorer is not supported

[10]For practical purposes, our implementation of the MSV JavaScript library employs the Script Tag Hack (http://javascript.crockford.com/script.html) in order to avoid the restrictions imposed by the Same Origin Policy in Web browsers. This means that measurements in our experiments additionally include the dynamic insertion of a script element in the DOM tree as well as script evaluation on response.

[11]Advanced Television Systems Committee (ATSC), IS-191 Relative Timing of Sound and Vision for Broadcast Operations.

[12]BBC R&D White Paper WHP185.

by our current implementation. While project members may not be a random selection of end-users, the experiments cover a wide range of devices, browsers and networks, as expected in multi-device scenarios.

7.2.1 Minimum Trans

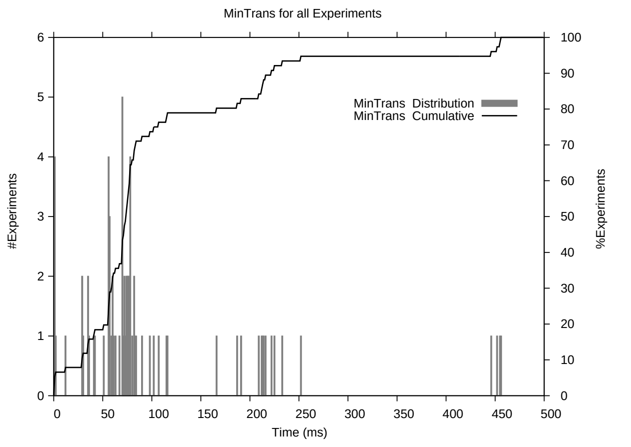

Figure 4: Distribution of minTrans for all experiments. The width of each bar is 1 ms. The height is the number of experiments that have minTrans values corresponding to the bar's position on the x-axis. The corresponding cumulative distribution illustrates the percentage of experiments that has minTrans less than X ms.

Each experiment includes one request-response sample that yields the minimum estimate for $trans$, which we call $minTrans$. This estimate is important for several reasons. It is a good basis for skew estimates and it provides a standard to which the other $trans$ estimates can be compared. In addition, it is indicative of the limitations of the network and the boundaries for synchronization error and accuracy.

Fig. 4 depicts the distribution of $minTrans$ for all experiments. Most clients have $minTrans$ values between 50 ms and 100 ms. The worst $minTrans$ recorded by any experiment is 456 ms. The cumulative distribution shows that about 80% of the experiments have $minTrans$ less than 120 ms.

These results let us speculate about the accuracy of regular MSV synchronization. In Cristian [1] a limit is given for the error of clock synchronization, $e = D(1 + 2p) - min$. In this formula D corresponds to $trans$, p is the relative clock drift and min is the shortest possible $trans$ across this network. We assume that clock drift is negligible over the short duration of our experiments. However, we cannot safely assume that $minTrans$ is a good estimate of min. For that to be the case, $minTrans$ would have to be derived from a much larger data set, spanning a much longer time interval. Instead, we simply assert that $e = trans - min$, where $0 < min \leq minTrans$. For the particular experiment sample where $trans = minTrans$ we have $e = minTrans - min$. In one extreme, where min is close to zero we have a maximal synchronization error close to $minTrans$. On the other hand, if $minTrans$ is close to min, the synchronization error becomes much smaller than $minTrans$. A conservative guess might be $min = minTrans/2$. If so, 80% of the experiments would have maximal error less than 60 ms. Note that this discussion relates to the synchronization error between client and server. The maximum synchronization error between two clients is the sum of the maximum error for each client.

7.2.2 Median and Maximum Trans

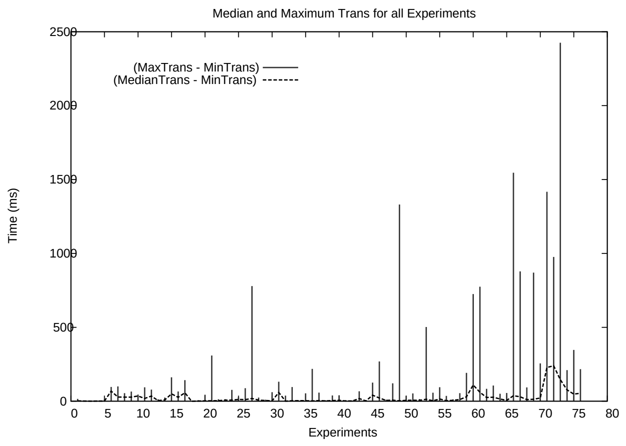

Figure 5: The dotted line shows the difference between medianTrans and minTrans, for each experiment. Similarly, the bars show the difference between maxTrans and minTrans. Experiments are sorted by minTrans, with lowest minTrans to the left.

Next we investigate the distribution of trans estimates for each experiment. Fig. 5 illustrates the *median* and *maximum* values derived from each 24 sample experiment. The difference between $medianTrans$ and $minTrans$ are shown by the dotted line. Ideally this line should be close to the x-axis, as sample distributions with the median close to the minimum value provide a good basis for probabilistic clock synchronization [1]. The median line peaks at 240 ms for an experiment with one of the highest $minTrans$. The bars represent the difference between $maxTrans$ and $minTrans$. With $medianTrans$ close to $minTrans$, $maxTrans$ is very often an outlier in the experiment. However, these outliers matter since they may be a noticeable source of asynchrony for MSV update synchronization. The biggest spike in this plot is 2.4 seconds. One outlier of 12.7 seconds is removed from the dataset. This value was recorded by one of the experiments with the lowest $minTrans$, so we suspect that this was due to exceptional behavior in the particular browser rather than a reflection of normal network behavior. Browser unreliability and inefficiency may play a part in other outliers as well. These results for $maxTrans$ suggest that introducing delay to *completely* mask variation in trans (see Section 5.2) would be impractical if update latency is critical. Note that update latency is only observable by the client that requested the update. Thus, in a broadcast scenario where all clients except the broadcaster should be denied access to the update primitive, long update latency might be quite acceptable.

7.2.3 Synchronization Startup

For each experiment, $minTrans$ is the best basis for clock

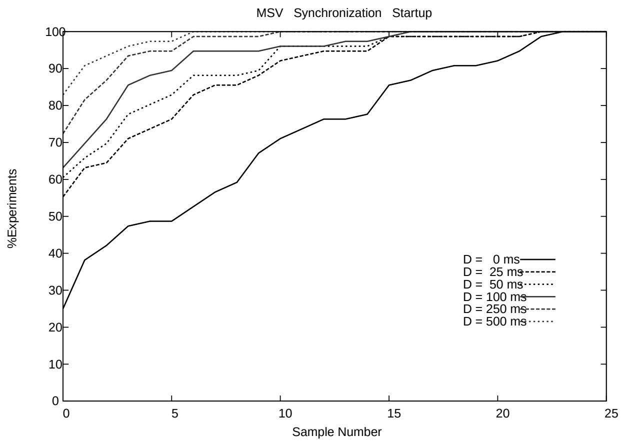

Figure 6: This illustrates how quickly experiments achieved an accurate estimate for minTrans. After X client-server exchanges, Y% of experiments obtained a trans estimate closer than D ms from minTrans. Each line represents a fixed value of D.

synchronization. We have investigated how quickly clients arrive at a trans estimate close to $minTrans$. Fig. 6 presents these results for six different measures of closeness, D. The line $D = 0$ reports when $minTrans$ was actually obtained. As expected, some clients obtain $minTrans$ early and some late. More surprisingly, in 25% of the experiments $minTrans$ was obtained in the very first sample. In about 85% of the experiments, a trans estimate closer than 25 ms from $minTrans$ was obtained after 10 samples. These results show that MSV clients are likely to obtain a decent estimate for trans quickly, for instance by issuing a modest batch of redundant requests as part of synchronization startup.

7.2.4 End-to-end Synchronization

Finally, we have conducted an experiment to verify the end-to-end accuracy of inter-client MSV synchronization, by videotaping multiple synchronized screens. The media presentation used in this experiment visualizes a binary clock[13]. As the underlying MSV is set in motion, the binary clock will tick simultaneously (ideally) on all screens. Synchronization error is quantified by mapping visual clock ticks to frame numbers in the captured video. We have performed this experiment once, with the MSV server in Tromsø and three screens at BBC R&D in London. The three screens are hosted by a Desktop, a Laptop and a Netbook. All three devices run Chrome browsers. The Desktop and the Laptop run Windows 7 and are connected to wired LAN, behind an HTTP proxy server. The Netbook runs Fedora 15 and is connected via wireless to a different network, with no HTTP proxy. The camera supports 30 frames per second. With this frame rate a synchronization error of 33.3 ms between Web browsers should imply an average difference of 1 frame in the captured video. The experiment involved recording a sequence of 47 consecutive clock ticks (5 seconds apart). The Desktop and the Laptop differed by 0 frames for 35 ticks and by 1 frame in the 7 remaining ticks. The Netbook differed from the Desktop by 0 frames for 15 ticks, by 1 frame for 31 ticks and by 2 frames for 1 tick. These results indicate that the average end-to-end inter-client synchronization error in this instance is a little less than 1 frame. This is close to lip-sync accuracy. Note that these results include error introduced by the Web browsers as they attempt to synchronize visual output with their local MSV proxies.

[13]Binary Clock demo at http://goo.gl/ugrj7

8. RELATED WORK

Timelines

Many concepts similar to the *Media State Vector* have already been discussed in the scientific literature: *timers, timelines, virtual timelines, dynamic variables, media clocks*. HyTime [4] introduced the notion of a time-based document and allowed media presentations to be specified declaratively in relation to an abstract timeline. In Athena Muse [5] timelines based on integer ranges were used to decouple channels of timed information. The project also proposed multi-dimensional media presentations. In TEMPO [6] a scripting language was introduced to control velocity and acceleration of computer animations. TBAG [7] used a set of interdependent *constrainables* to control 3D animations under continuous motion. Nsync [8] focused on prediction and implemented a media *clock* like the MSV, but without support for acceleration. In Flash ActionScript[14] a media presentation is constructed by binding event handlers to a navigable timeline.

The MSV can be viewed as a refinement of these concepts. It is expressive, yet compact and easy to use. It is explicitly defined as an independent object, and does not prescribe a particular execution model. Most importantly though, the MSV is motivated differently. The above projects use the timeline concept as a basis for construction and execution of a single media experience, not as a synchronization mechanism between components of a distributed media presentation. The only exception is TBAG [7] where synchronization of *constrainables* across the Internet is discussed. These projects complement our thesis by attesting to the claim that media presentations can be modeled as functions of motion.

Distributed Media Synchronization

Synchronization of distributed media presentations has been much explored in the context of continuous AV streams and multicast networks. Important applications types include video conferencing, screen sharing and Web-cast. In MODE [9] video and non-continuous content is multicast and synchronized between multiple nodes, relative to predefined reference points. ACME [10] and Tactus [11] focus on the synchronization of multiple source streams into one single presentation. In ACME a *logical time system* is developed for this purpose. Group synchronization over multicast networks [12, 13, 14] usually depends on introducing delay at the destinations to mask differences in network latency. Clock synchronization is either assumed or implemented as part of the solution.

In contrast, the MSV approach advocates decoupling of motion from content. This implies that motion and content can be communicated across different channels and technologies. The MSV approach makes motion available at the destinations and allows clients to control content delivery, by pulling in video streams according to client-side policies.

[14]http://www.adobe.com/

As long as the client can map MSV behavior to video service requests, any stream-based transport mechanism can in principle be used for the video content. A second implication is that media presentations without stream-based media may also be synchronized using the same mechanism. In fact, this challenges established definitions for distributed multimedia. For instance, Blakowski *et al.* [15] distinguish time-dependent and time-independent media and requires multimedia systems to include at least one time-dependent media type. This definition must be revised, unless the MSV itself is considered a media type. Finally, by addressing the issue of media synchronization at application-level rather than the transport-level, the MSV approach becomes an instance of the end-to-end argument [16].

Second Screen Scenarios

Second screen scenarios usually involve synchronization of secondary devices (e.g., smart phones and tablet PCs) to a primary device (e.g., TV). Disney Second Screen[15] allows extra material for movies by synchronizing iPads to a Blu-ray player across the local network. Orchestrated Media is a recent BBC project that investigates synchronization of secondary devices to broadcast TV using solutions based on digital watermarking [3] for inter-device synchronization. Howson *et al.* [17] also recognizes the need for a distributed event timeline, decoupled from media format and transport protocol. However, despite being logically independent, timeline events are still bundled and transmitted with the content. Kahmann *et al.* [18, 19] share our vision of a unified approach to collaboration and secondary screen scenarios, and focus on session management for collaborative media streaming. A session object is similar to the MSV in the sense that both encapsulate media control and session membership. However, the approach is limited to streamed media on the home network.

The MSV approach challenges current solutions for second screen scenarios by suggesting that the local problem of device synchronization may be solved remotely. In doing this, the MSV approach bypasses many problems associated with local network synchronization; device discovery, time-consuming switching between network carriers, multiple networks, inter-device communication protocols, etc. Furthermore, it supports symmetric navigation, view autonomy and instance-view independence by avoiding an asymmetric master-slave relationship between devices. It also removes the need for local hosting of synchronization services and it simplifies content production for secondary screen scenarios, essentially by making it a special case of Web publishing. In short, the MSV approach eliminates the distinction between home and away.

Synchronization in Web browsers

SMIL [20] is a framework for media synchronization in Web browsers. SMIL aims to combine the benefits of declarative authoring typical of video-centric synchronization models, with the flexibility of timing models typical for animations and graphics. The framework allows DOM access to control speed and acceleration. Unlike SMIL, MSV synchronization does not address authoring. Still, we argue that various authoring frameworks, including declarative frameworks like SMIL, could utilize the MSV concept.

[15]http://disneysecondscreen.go.com

The introduction of HTML5 made it practical to exploit the timeline of the HTML5 video element as a basis for synchronized media presentations in Web browsers. We were inspired by LIMO, a JavaScript framework made by BBC R&D as part of P2P-Next. LIMO eased construction of such lightweight, interactive video-based media presentations. More recently Popcorn.js[16] has open sourced similar functionality and attracted an enthusiastic user base. Both projects demonstrate the ability of modern Web browsers to weave together lightweight media presentations in real time, in response to the motion of the video element. The MSV approach can potentially increase the utility of these frameworks, by removing the dependence on the video element, and by allowing them to become authoring frameworks for multi-screen media presentations.

SMIL, LIMO and Popcorn all address synchronization in a single web-document. In contrast, Wijnants *et al.* [21] address synchronous media sharing *(sMS)* on the Web. A framework is presented that allows regular web-browsers on heterogeneous devices to concurrently navigate shared media presentations. Media navigation is restricted to discrete jumps, making the solution appropriate for slideshow navigation, but not for precise synchronization of continuous media. Motion and content are weakly decoupled, as sMS session records (motion) and RSS feeds (content) are independently available. However, sMS session records include references to RSS feeds. Furthermore, the architecture suggests that content and motion must be hosted by the same sMS server. In contrast, the MSV approach imply that synchronization servers are content agnostic, and that management of the mapping between content and motion can be shifted onto clients, or other application servers.

9. CONCLUDING REMARKS

The primary objective of this paper is to introduce the concept of the *Media State Vector (MSV)* and to demonstrate its utility as a general basis for synchronization of media presentations. The secondary objective is to suggest MSV synchronization as a foundation for cloud-based services offering generic, multi-device, media navigation across the Internet. By meeting all the requirements for cloud-based media presentations we claim that both objectives are satisfied.

Furthermore, the evaluation performed by current Web browsers demonstrates the feasibility and potential of this approach. We argue that maximum synchronization errors of less than 120 ms (for 80% of the test clients) is already adequate for a wide range of applications. Observed inter-device synchronization error of about 33 ms across Europe shows that MSV synchronization may be precise, even without any optimizations. With further reference to our primitive, application level implementation of clock synchronization, we claim that MSV synchronization accuracy can only be improved.

We find the MSV to be a powerful concept, supported by the following key ideas: 1) linear motion as a unified approach to media navigation, 2) decoupling of motion from content and presentation, 3) media presentations implemented as functions of motion and content, 4) synchronization of media by synchronization of motion, and 5) synchronization of motion across the Internet.

[16]http://popcornjs.org/

Finally, we are confident that the MSV concept redefines media presentations by removing significant limitations, and that it has wide utility in new as well as existing media applications.

Acknowledgements

The authors would like to thank members of the LIMO project at BBC R&D for bringing to our attention the ability of modern Web browsers to synchronize media to the progress of a video element. W. Bamford, Lancaster University is acknowledged for his contributions to LIMO as part of P2P-Next. We would also like to thank R. van Renesse (Cornell University), Y. Raimond and D. Tinley (BBC), M.K. Skadsem, D. Stødle, T. Grydeland and Y. Larsen (Norut) for providing valuable feedback to this work. Mr. Larsen is especially acknowledged for his assistance in naming the MSV concept.

10. REFERENCES

[1] F. Cristian, "Probabilistic clock synchronization," *Distributed Computing*, vol. 3, pp. 146–158, 1989.

[2] D. Mills, "Internet time synchronization: the network time protocol," *IEEE Transactions on Communications*, vol. 39, no. 10, pp. 1482–1493, Oct 1991.

[3] R. van Schyndel, A. Tirkel, and C. Osborne, "A digital watermark," in *Image Processing, 1994. Proceedings. ICIP-94., IEEE International Conference*, vol. 2, Nov 1994, pp. 86–90.

[4] S. Newcomb, N. Kipp, and V. Newcomb, "The hytime: hypermedia/time-based document structuring language," *Communications, ACM*, vol. 34, pp. 67–83, November 1991.

[5] M. Hodges, R. Sasnett, and M. Ackerman, "A construction set for multimedia applications," *Software, IEEE*, vol. 6, no. 1, pp. 37–43, Jan 1989.

[6] L. Dami, E. Fiume, O. Nierstrasz, and D. Tsichritzis, "Temporal scripting using tempo," in *Active Object Environments (ed. D. Tsichritzis)*, Centre Universitaire d'Informatique, Université de Genève, 1994.

[7] C. Elliott, G. Schechter, R. Yeung, and S. Abi-Ezzi, "Tbag: a high level framework for interactive, animated 3d graphics applications," in *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH '94. New York, NY, USA: ACM, 1994, pp. 421–434.

[8] B. Bailey, J. Konstan, R. Cooley, and M. Dejong, "Nsync - a toolkit for building interactive multimedia presentations," in *Proceedings of the sixth ACM international conference on Multimedia*, ser. MULTIMEDIA '98. New York, NY, USA: ACM, 1998, pp. 257–266.

[9] G. Blakowski, J. Hübel, U. Langrehr, and M. Mühlhäuser, "Tool support for the synchronization and presentation of distributed multimedia," *Computer Communications*, vol. 15, no. 10, pp. 611–618, 1992.

[10] D. Anderson and G. Homsy, "A continuous media i/o server and its synchronization mechanism," *Computer*, vol. 24, no. 10, pp. 51–57, Oct 1991.

[11] R. Dannenberg, T. Neuendorffer, J. Newcomer, D. Rubine, and D. Anderson, "Tactus: toolkit-level support for synchronized interactive multimedia," *Multimedia Systems*, vol. 1, pp. 77–86, 1993.

[12] I. Akyildiz and W. Yen, "Multimedia group synchronization protocols for integrated services networks," *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 1, pp. 162–173, Jan 1996.

[13] J. Escobar, C. Partridge, and D. Deutsch, "Flow synchronization protocol," *IEEE/ACM Transactions on Networking*, vol. 2, no. 2, pp. 111–121, Apr 1994.

[14] Y. Ishibashi and S. Tasaka, "A group synchronization mechanism for live media in multicast communications," in *Global Telecommunications Conference, 1997. GLOBECOM '97., IEEE*, vol. 2, Nov 1997, pp. 746–752.

[15] G. Blakowski and R. Steinmetz, "A media synchronization survey: reference model, specification, and case studies," *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 1, pp. 5–35, Jan 1996.

[16] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Transactions on Computer Systems*, vol. 2, pp. 277–288, Nov 1984.

[17] C. Howson, E. Gautier, P. Gilberton, A. Laurent, and Y. Legallais, "Second screen tv synchronization," in *2011 IEEE International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, Sep 2011, pp. 361–365.

[18] V. Kahmann and L. Wolf, "Collaborative media streaming in an in-home network," in *2001 Distributed Computing Systems Workshop*, Apr 2001, pp. 181–186.

[19] V. Kahmann, J. Brandt, and L. Wolf, "Collaborative streaming in heterogeneous and dynamic scenarios," *Communications of the ACM*, vol. 49, no. 11, pp. 58–63, Nov 2006.

[20] P. Schmitz, "Multimedia meets computer graphics in smil2.0: a time model for the web," in *Proceedings of the 11th international conference on World Wide Web*, ser. WWW '02. New York, NY, USA: ACM, 2002, pp. 45–53.

[21] M. Wijnants, J. Dierckx, P. Quax, and W. Lamotte, "synchronous mediasharing: social and communal media consumption for geographically dispersed users," in *Proceedings of the 3rd Multimedia Systems Conference*, ser. MMSys '12. New York, NY, USA: ACM, 2012, pp. 107–112.