Video BenchLab: An Open Platform for Realistic Benchmarking of Streaming Media Workloads

Patrick Pegus II Emmanuel Cecchet Prashant Shenoy
University of Massachusetts Amherst
{ppegusii,cecchet,shenoy}@cs.umass.edu

ABSTRACT

In this paper, we present an open, flexible and realistic benchmarking platform named *Video BenchLab* to measure the performance of streaming media workloads. While Video BenchLab can be used with any existing media server, we provide a set of tools for researchers to experiment with their own platform and protocols. The components include a MediaDrop video server, a suite of tools to bulk insert videos and generate streaming media workloads, a dataset of freely available video and a client runtime to replay videos in the native video players of real Web browsers such as Firefox, Chrome and Internet Explorer. We define simple metrics that are able to capture the quality of video playback and identify issues that can happen during video replay. Finally, we provide a Dashboard to manage experiments, collect results and perform analytics to compare performance between experiments.

We present a series of experiments with Video BenchLab to illustrate how the video specific metrics can be used to measure the user perceived experience in real browsers when streaming videos. We also show Internet scale experiments by deploying clients in data centers distributed all over the globe. All the software, datasets, workloads and results used in this paper are made freely available on SourceForge for anyone to reuse and expand.

Categories and Subject Descriptors

D.2.5 [**Testing and Debugging**]: Testing tools, .D 2.8 [**Metrics**]: Performance measures.

General Terms

Measurement, Performance, Experimentation.

Keywords

Benchmarking, Video, Streaming, Web browsers.

MMSys '15, March 18 - 20, 2015, Portland, OR, USA
Copyright is held by the owner/author(s).
Publication rights licensed to ACM.
ACM 978-1-4503-3351-1/15/03…$15.00
http://dx.doi.org/10.1145/2713168.2723145

1. INTRODUCTION

In just over two decades, streaming media has become ubiquitous in our digital lives. Ordinary users are now able to create streaming media content using ordinary smartphones or inexpensive digital camcorders. Online services such as YouTube, Netflix and live streaming services such as Justin.tv provide a broad variety of content for our entertainment. Today online video is often viewed from a range of mobile devices such as phones or tablets as well as larger displays such as TVs connected to Internet streaming devices.

Although streaming media has gone mainstream, it continues to raise new opportunities and research challenges, and multimedia systems researchers have been working on a range of topics from DASH protocols, caching and content distribution, and clustered media servers. New topics such as the use of cloud computing and mobile computing in the context of streaming have emerged in recent years. While research on multimedia systems continues to flourish, researchers face several hurdles in carrying out experimental aspects of this research. There is a dearth of streaming media benchmarking tools to measure the performance of servers, clients and the network; while a few commercial benchmarking tools exist [26], they present difficulties for free research use or may not be amenable to modifications. The dataset track at the ACM Multimedia Systems conferences has been instrumental in gathering numerous open datasets for research use [1], but realistic performance evaluation involves more than datasets and traces—tools to easily set up experiments, generate workloads, and gather results are needed, especially when experiments run on myriad types of client devices and remote servers or cloud systems. In the absence of such tools, a researcher is left with the option of using homegrown tools or cumbersome manual experimentation.

The BenchLab project seeks to address these limitations by designing an open, freely-available platform for realistic benchmarking of servers applications. While BenchLab was initially designed to support web-based applications and services (e.g., multi-tier web applications accessed from browser-based clients), in this paper we describe *Video BenchLab*, an enhanced platform that provides similar functions for streaming media servers and protocols accessed via browser-based players. Our overall goals are to provide an open, flexible and realistic environment to generate and inject client streaming workloads onto servers to enable careful experimental evaluation of the performance of streaming servers, clients and network protocols. Towards this end, Video BenchLab uses real web browsers running real video players to request HTTP streaming content from the server. The platform supports desktop-based, mobile phone-based and tablet-based clients for generating workloads. A key goal of Video BenchLab is to enable automation for running complex experiments where clients, servers or both may be distributed or running on remote machines such as cloud servers; an experiment comprises a set of clients that are controlled remotely by the platform and provided with a video request trace that they then inject on the specified server or set of servers. The content is streamed to HTML5 players supported by modern browsers and a range of statistics are gathered and uploaded to a central database. Real user behavior when watching video content such as pausing, seeking, changing video quality or clicking on related videos can be simulated as part of the trace replay. BenchLab provides the ability to inject requests onto real servers (e.g., YouTube) and also provides a synthetic server backend based on MediaDrop that emulates a YouTube-like system on small scale. Our goal is to enable a range of performance evaluation experiments related to server design, streaming protocol performance, network performance and client-side performance. Finally, Video BenchLab includes new tools that permit analysis of results across different runs of an experiment. We believe that the open nature of the BenchLab platform makes it an attractive choice for multimedia systems researchers for use in their own research and experimentation.

In designing and implementing Video BenchLab, this paper makes the following contributions:

1) *Platform:* Video BenchLab brings the many benefits of the original BenchLab platform to streaming media performance evaluation. Like BenchLab, Video BenchLab is an open platform constructed using open-source components; researchers are free to use the entire platform for running and automating experiments, or any subset of BenchLab's tools.

2) *Server backend:* We design a video server appliance that is based on the open MediaDrop platform that seeks to emulate a YouTube-like system that enables users to upload content, view uploaded content as well as related videos. MediaDrop is a content manager rather than a streaming server and we envision that researcher will be able to "plug in" their own streaming protocols into the platform (e.g., a new DASH protocol) when using Video BenchLab in their research. A researcher using Video BenchLab is not restricted to using our server backend—indeed any other HTTP streaming server could be used as a backend, and it is even possible to use existing services such as YouTube as the server for experimentation.

3) *Video dataset and tools:* We provide tools for researchers to efficiently upload any video dataset of their choice into our MediaDrop-based server backend for their experiments. In addition, we provide a virtual machine appliance where MediaDrop is pre-loaded with a set of videos that are freely available without licensing restrictions; both desktop and mobile versions of these videos are available to enable flexible experimentation. While Video BenchLab is able to replay *any* request trace that is provided by a researcher, we also provide tools to create custom request traces that are tailored to the content loaded on the server, with the ability to vary statistical properties of the traces.

4) *Experimental Use Cases:* Finally we present numerous use case scenarios to illustrate how a researcher might use Video BenchLab in their research by showing what types of experiments can be run, and how these results might be used to understand the behavior of the client, server or the network. We also show how Video BenchLab enables experimental results to be gathered, analyzed and compared and used for what-if analyses.

The rest of this paper is structured as follows. We present Video BenchLab and its different components in section 2. We describe different usage scenarios and experimental results in section 3. Section 4 discusses related work and we conclude in section 5.

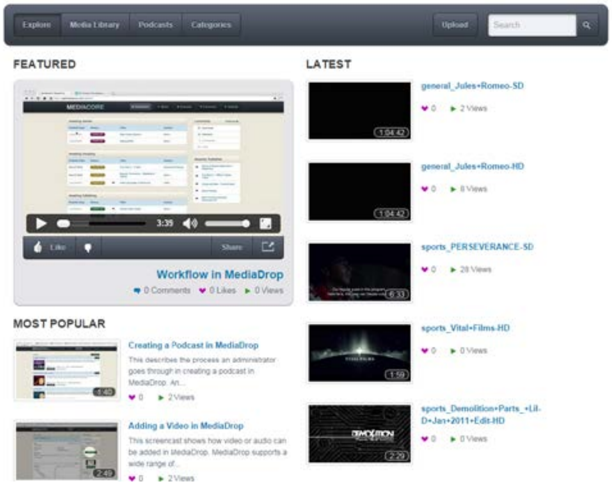
2. VIDEO BENCHLAB

Video BenchLab provides multiple components and tools for researchers to experiment with media streaming environments: a streaming server virtual appliance (section 2.1), video data sets and workload generators to exercise these datasets (section 2.2), a client runtime to play videos in existing Web browsers using their native HTML5 players (section 2.3) and a Web Application hosting a database to define experiments, collect results and provide analytics (section 2.4).

2.1 MediaDrop Virtual Appliance

While Video BenchLab can be used with any existing video streaming service, there is no readily available virtual appliance of realistic video streaming servers that can be used by researchers to perform media streaming experiments. Recently, in-browser HTML 5 video playing has become standard since the switch of all major players to HTTP streaming [19]. To provide a relevant benchmark, we wanted a redistributable open source platform supporting HTML5 video streaming that also included social media features such as comments, view counts and likes.

MediaDrop [12] is a popular open source video platform that supports HTML5 and Flash videos on most platforms. Videos can be imported from many sources including YouTube or Amazon S3, and replayed in any desktop browser or mobile platform. Like modern video streaming platforms, MediaDrop offers content management, statistics and popularity ranking and social media aspects such as comments, Twitter and Facebook integration (see Figure 1).



This describes the process an administrator goes through in creating a podcast in MediaDrop. An...

This screencast shows how video or audio can be added in MediaDrop. MediaDrop supports a wide range of...

Figure 1. Screenshot of the MediaDrop server home page

A number of HTML5 video players are available and offer various tradeoffs in terms of platform support and integration in Web pages and browsers [3]. We decided to rely on the browser native HTML5 video player for our MediaDrop VM. MediaDrop can easily be reconfigured to use Javascript based players to experiment with various streaming libraries too. We note that as a video content management system, MediaDrop is not tied to any specific streaming server; it supports basic HTTP streaming and it is possible to use other HTTP-based streaming servers and protocols with MediaDrop for additional experimentation.

We have built an Amazon EC2 virtual machine image with a complete installation of MediaDrop ready to use. This virtual appliance is publicly available on EC2. We also provide a script that can automatically build a new MediaDrop VM from scratch by downloading the required software and configuring it. This allows researchers with their own private cloud to build custom VMs for their cloud platform.

Importing videos through the MediaDrop Web interface is a multistep process that involves a user uploading a video and an administrator approving the video for it to be available for others to stream. This process is too cumbersome for bulk video inserts or large dataset creations. We provide tools to perform bulk inserts of videos directly in the MediaDrop database to make them readily available with their thumbnails. Videos can also be converted between multiple formats during the import process using the provided libav tools [15]. Our MediaDrop appliance comes with a set of freely redistributable videos. The content of the dataset is described in section 2.2.

We also provide the Ganglia monitoring tools [5] to report on cpu, memory, disk and network utilization if the cloud infrastructure does not already report these metrics. When network throttling is needed, we rely on the Linux traffic controller hierarchical token bucket (tc-htb) to limit the MediaDrop server bandwidth.

2.2 Video datasets, tools, and workloads

While Video BenchLab allows a researcher to upload any video dataset into the MediaDrop appliance, for ease of use, we provide an initial video dataset for research use. All videos in our dataset were obtained from a set of videos on Vimeo [25] that are distributed under a Creative Commons Attribution and thus can be used and modified by researchers without licensing restriction. Table 1 summarizes our video dataset, which contains videos on topics such as news, general entertainment and sports. Videos are classified in 3 categories by their duration and are all encoded in WebM format using the VP8 video and Vorbis audio codecs. Longer videos are not always necessarily bigger in file size depending on the effectiveness of the compression on the video and audio streams.

Table 1. Video dataset specifications

Category	# of videos	Duration	Video size
Small	20 (10SD-10HD)	0-5 min	2.9-68MB
Medium	8 (4SD-4HD)	5-10 min	20MB-73MB
Large	5 (3SD-2HD)	10min – 1H+	18MB-203MB

Typical BenchLab workload traces include a unique request id, a timestamp indicating when the request must be played, a client id identifying the browser, the URL to visit as well as optional interactions with Web elements in the visited page. The trace format for video workloads extends the original format by adding video manipulation commands. The supported commands are:

- `play`: play the video from the current position
- `pause`: pause the video at the current position
- `change_quality(`*`quality`*`)`: changes the quality of the video (player specific, e.g. in native HTML5 implemented as pause/load new media file/seek to previous current position/resume, in YouTube implemented as a call to setPlaybackQuality). *`quality`* is one of the pre-defined values `lowest`, `highest`, `lower` or `higher`, which changes to the lowest, highest, next lower available and next higher available quality, respectively.
- `skip_ads`: skip the in-stream advertisement (player specific, currently only available for YouTube).
- `quit`: end the playing right now and proceed to the next entry in the trace
- `seek(`*`video_position`*`)`: seek the video to the given position in seconds since the beginning of the video. The position is defined by a numerical value or a formula using the video `length` and `current_position` provided by the BCR. Example:
    - Seek to 1 minute into the video: `seek(60)`
    - Seek to the middle of a video: `seek(length/2)`
    - Jump 30 seconds: `seek(current + 30)`
- `wait_for(`*`timeout,[video_position]`*`)`: wait for the timeout to expire or the current video position to reach the given value, whichever comes first. The `wait_for` command should always be followed by another command. The timeout can either be a relative value in seconds or an absolute timestamp. The *`video_position`* parameter is defined in a similar way as the seek command:
    - Wait for the video to reach 1 minute playback within 2 minutes: `wait_for(120, 60)`
    - Wait for the video to reach the middle of the playback with a maximum 20% delay: `wait_for((length/2)*1.2, length/2)`
    - Wait until the given timestamp before executing the next command: `wait_for('2015-01-16 12:30:10.5')`
    - Wait until the video ends: `wait_for(0,length)`

The default script that just plays a video until its end is as follows (note that `player_type` is either `html5` or `youtube`):

```
  VideoBenchLab={player_type:html5,
    commands:{play, wait_for(0,length), quit}}
```

Multiple browsers can be synchronized by using the timestamp field in the trace. Recall that the trace format is: `<request id>,<client id>,<timestamp>,<URL>,<parameter>`

The following trace starts 3 clients at a 10 seconds interval, each client going first to the YouTube homepage and 5 seconds later playing a video (URLs are abbreviated):

```
1, 1, 2015-01-16 00:00:00.0, http://youtu.be,null
2, 1, 2015-01-16 00:00:05.0, http://youtu.be/…,VideoBenchLab={…}
3, 2, 2015-01-16 00:00:10.0, http://youtu.be,null
4, 2, 2015-01-16 00:00:15.0, http://youtu.be/…,VideoBenchLab={…}
5, 3, 2015-01-16 00:00:20.0, http://youtu.be,null
6, 3, 2015-01-16 00:00:25.0, http://youtu.be/…,VideoBenchLab={…}
```

We have created a trace generator tool that can generate workloads traces following predefined distributions such as a Zipf distribution [18][9]. The generator requires a list of video metadata objects in JSON format containing the video URL, duration, and a list of URLs of web pages containing related videos. We provide a tool that automatically extracts that information from the MediaDrop database to feed it to the workload generator. By suitably choosing parameters for the workload generation tools, a researcher can construct traces with different number of concurrent users accessing videos from the server with different popularity skews. Of course, a researcher is not restricted to using our traces and may use any other trace of their choosing for running experiments (so long as the traces adhere to the format used by BenchLab clients).

2.3 Video BenchLab Client Runtime

A central contribution of Video BenchLab is the ability to replay traces through real Web browsers. Major companies such as Google and Facebook already use open source technologies like Selenium [19] to perform functional testing using real web browsers. These tools automate a browser to follow a script of actions, and they are primarily used for checking that a Web application's interactions generate valid HTML pages. We argue that the same technology can also be used for video performance benchmarking.

2.3.1 *Architecture overview*

The Video BenchLab Client Runtime (BCR) extends the Selenium framework with functionalities to download a video trace, replay it via a real web Browser by issuing HTTP requests for video, record streaming performance statistics for each page and upload the results at the end of the replay. Unlike traditional load injectors that work at the network level, replaying through a Web browser accurately performs all the complex interactions between the browser and the server. When playing videos, the unmodified native player or plugin extensions of the browser are used giving insights on real world performance on any platform or software combination. This level of complexity could not be approximated by simulators especially given the wide variety of factors affecting performance. Through the browser, the BCR captures the real user perceived latency including network transfer, page processing and rendering time.

Most desktop browsers include debugging tools such as Firebug for Firefox or the developer tools for Chrome that are able to capture the timeline of all browser interactions while pages are being loaded. This data can usually be stored into a HTTP Archive (HAR) file [11]. We use the open source BrowserMob proxy [29] developed by WebMetrics to record network activity in HAR files. Figure 2 gives an overview of the architecture of the BCR with a Firefox browser as an example. We also support the Chrome and Internet Explorer browsers.

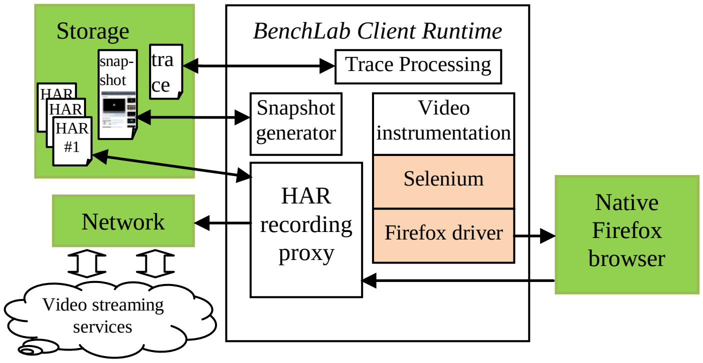


Figure 2. Video BenchLab Client Runtime (BCR) architecture

In order to benchmark HTML5 video, the BCR executes JavaScript at regular time intervals on the video web page to collect performance metrics (see section 2.3.2). The script instructs the browser to record video metadata, including MIME type, resolution, and duration, then log time, the video specific metrics, and, if supported by the player, when the video has stalled. The video commands included in the trace file are executed according to the player type (currently native HTML5 and YouTube are supported). By default, the BCR plays videos entirely and then saves the page HTML source with the log of the video replay. These results are uploaded automatically to the BenchLab Dashboard database at the end of the experiment.

Additionally the BCR can record screen snapshots of rendered pages. This data is automatically uploaded at the end of the experiment and can be visualized in the dashboard. As shown on Figure 3, the screen snapshots can also capture errors reported by the browser. Errors such as connection reset or invalid source (unsupported video format) can be easily detected this way.

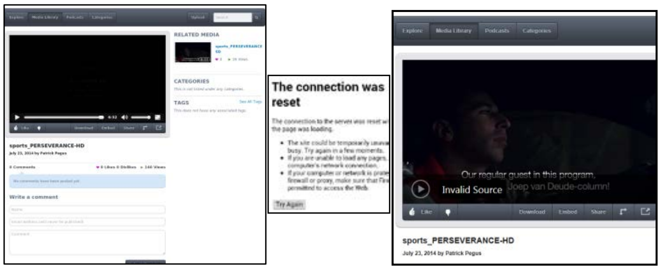


Figure 3. Example of screen snapshots taken during experiments with MediaDrop (left: normal execution, middle: network error, right: unsupported video format).

2.3.2 *Performance metrics*

The general performance statistics collected by the BCR during an experiment are stored in a standard HTTP archive (HAR) format, and include the following metrics: DNS resolution time, connection establishment time, request failure/success/cache hit rate, send/wait/receive time on network connections, overall page loading time including Javascript execution and rendering time. Figure 5 shows an example of a HAR capture during one of our video experiment with MediaDrop.

For videos, the BCR collects current time, video position, resolution, buffer start and end positions, and, if supported by the player, when the video has stalled. As each player might store these values differently, the data collection code is specific to each video player. We have implemented data collection for the native standard HTML5 player found in every browser and also for the YouTube player. Data collectors can be easily adapted to any custom Javascript video player by simply implementing an interface that describes the values to collect.

We use the collected data to generate several metrics to assess video performance as perceived by the user. We record the current video position V_c and the current time T at regular intervals (default is 1 second) during video playback. If both V_c and T progress at the same pace, the video is playing smoothly.

We define a metric called *lag* defined by $max(0, \Delta T - \Delta V_c)$. When a video lags or stop playing, V_c moves more slowly than T. While the clock timestamps have a milliseconds precision we usually filter out any *lag* value below 100ms.

Similarly we define a metric called *skips* that is calculated by $max(0, \Delta V_c - \Delta T)$. When the current video position moves faster than the clock time, it means that the player is skipping parts of the video. This condition can happen if the stream is corrupted or the player cannot play the stream fast enough and needs to skip frames.

We also report the video buffer size by recording the buffer start position B_s and end position B_e, buffer size being B_e-B_s. Usually players will try to cache the entire video to be able to do in-memory replays of parts of the video that have already been played. This means that B_s will typically not change and only B_e

will increase at the rate of the network download speed (though this might not always be true as video buffering might be decoupled from network transfers).

2.4 Running experiments with the BenchLab Dashboard

The BenchLab Dashboard is a key component of our platform that is used to setup and automatically run performance experiments as well as gathering experimental results. Thus, it is the central repository for experiments. It is built as a Java Web application that can be deployed in any Java Web container such as Apache Tomcat. The BenchLab Dashboard provides a Web interface to interact with experimenters that want to create experiments using an arbitrary number of browsers and servers. The Dashboard gives an overview of the browsers currently connected, the experiments (created, running or completed) and the Web traces that are available for replay.

Video workload trace files are uploaded by the experimenter through a Web form and stored in the Dashboard database. The BenchLab Dashboard does not deploy, configure or monitor any server-side software. There are a number of deployment frameworks (Gush, WADF, JEE, .Net deployment service, etc) and monitoring tools (Ganglia and fenxi are popular choices) that users can choose from depending on their preferences.

Anyone can deploy a BenchLab Dashboard and therefore build his or her own experiment repository. An experiment defines what trace should be played and how. The user defines how many browsers should replay the sessions with eventual constraints (specific platform, version, location…). The experiment can start as soon as enough clients have registered to participate in the experiment. The Dashboard does not deploy client web browsers, rather it waits for the browsers to connect and its scheduler assigns them to experiments.

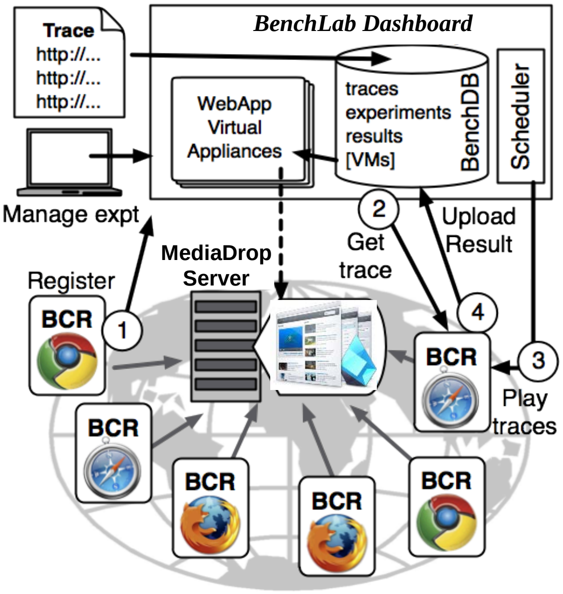


Figure 4. Video BenchLab experiment flow overview

Figure 4 gives an overview of the BenchLab components and how they interact to run an experiment. The Video BenchLab Client Runtime (BCR) starts and controls the native Web browser on the client machine. On startup, the BCR connects the browser to a BenchLab Dashboard (step 1 in Figure 4). When the browser connects to the Dashboard, it provides details about the exact browser version and platform runtime it currently executes on as well as its IP address and location if available. If an experiment needs this browser, the Dashboard redirects the BCR to a download page where it automatically gets the trace for the session it needs to play (step 2 in Figure 4). The BCR stores the trace on the local storage and makes the Web browser regularly poll the Dashboard to get the experiment start time. There is no communication or clock synchronization between BCRs, they just get a start time as a countdown in seconds from the Dashboard that informs them 'experiment starts in x seconds' through a Web form. The status of browsers is recorded by the Dashboard and stored in a database.

When the experiment start time has been reached, the BCR plays the trace through the Web browser monitoring each interaction (step 3 in Figure 4). For each URL visited, the BCR records the Web and video performance metrics described in section 2.3.2. The results are uploaded to the Dashboard at the end of the experiment (step 4 in Figure 4). The Dashboard provides a number of ways of visualizing data and comparing results between experiments. The entire database with experiment configuration, traces and results can also be easily exported to be shared with other researchers.

It is possible to crowd-source experiments by defining an experiment with a large number of users (the trace must contain at least 1 session per user). As soon as the number of users is reached, the Dashboard automatically triggers the start of the experiment. The Dashboard is able to process and display partial results in case clients failed during the experiment. We show Internet scale experiments in section 3.3 that rely on that feature. Browsers are deployed in various data centers and experiments start automatically as soon as enough browsers have connected to the Dashboard.

2.5 Implementation and Platform Availability

As noted earlier, Video BenchLab is an open platform that is designed for benchmarking streaming media workloads. Hence, all components of the Video BenchLab software suite (see Table 2) are freely available under an open source license.

Table 2. Summary of tools in the Video BenchLab suite

Tool	Description	Platforms
MediaDrop	Video streaming server	Linux
mediaload	Bulk video import for MediaDrop	Linux/python
metadump	Video metadata export	Linux/python
loadgen	Video workload generator	Linux/python
BCR	Video BenchLab Client Runtime	Firefox+Chrome on Linux/Windows, IE on Windows
HTML5 collector	BCR data collector for HTML5 video players	All browsers
YouTube collector	BCR data collector for YouTube video player	All browsers
Dashboard	Experiment management, result db and analytics	Java Web container (e.g. Tomcat)

These components are available individually and as a packaged platform to permit flexibility in how they are used for experimentation. They are available for download from SourceForge at http://sf.net/projects/benchlab. We have also published our experimental Dashboard database with all the results contained in this paper on SourceForge. To help with reproducibility of experimental results, anyone can import this database in their own Dashboard and rerun the experiments. We have also provided Amazon EC2 images (AMIs) for MediaDrop, BCR and Dashboard so that anyone can reproduce the experiments conducted in this paper. Additional details of our platform are available from the BenchLab project page at http://lass.cs.umass.edu/projects/benchlab.

3. EXPERIMENTAL USE CASES

In this section, we present a series of experiments that illustrate "use cases" and the types of experiments that can be run with BenchLab using our MediaDrop appliance. Section 3.1 describes our experimental setup and methodology. We show how video specific metrics gathered by our platform can be used to determine the user perceived quality of experience in section 3.2. Finally we showcase Internet scale experiments that can be automated and run using our platform in section 3.3.

3.1 Experimental setup and methodology

To demonstrate the type of experiments that can be run using Video BenchLab, we deployed virtual machines on Amazon EC2 for our MediaDrop implementation, BenchLab Dashboard and all the BCRs (BenchLab Client Runtime) running the various browser implementations. All instances used were of type m1.small deployed in the North Virginia data center (US East Coast) unless indicated otherwise. Internet scales experiments have additional deployments in other EC2 data centers.

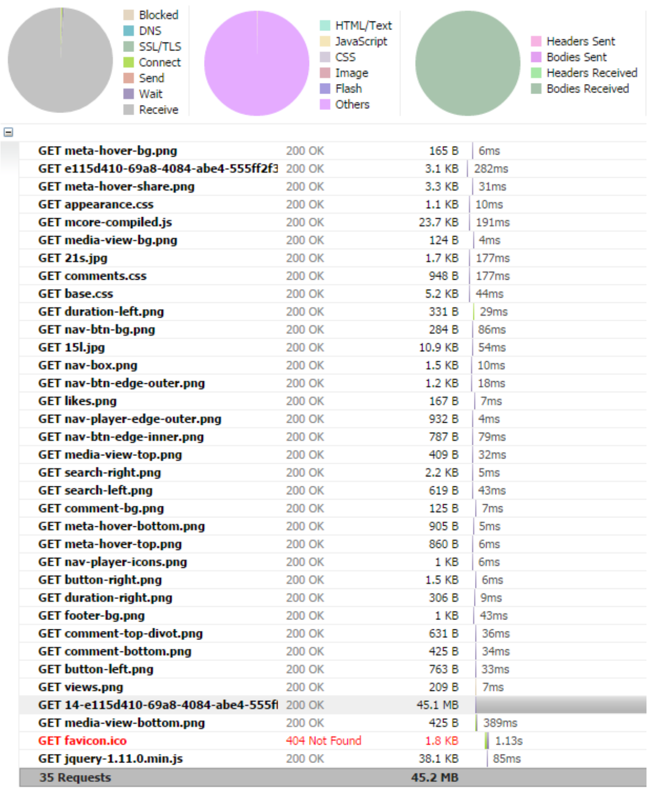
The browsers used in the experiments are: Firefox 30 on Ubuntu Linux 14.04 x86_64, Chrome 36.0.1985.125 on Ubuntu Linux 14.04 x86_64 and Internet Explorer 11 on Windows 7. BenchLab client and Dashboard software was version 2.2 (including Selenium 2.43.1, Chrome driver v2.9 and IE driver v2.43 32 bits). MediaDrop version 0.10.3 was used with the HTML5 player instead of the default Flash based player.

Experiments involving a single browser were performed in isolation (only one browser at a time streaming a video from the server). Experiments involving multiple browsers were synchronized on start by the BenchLab Dashboard. In each trace, we use the default script that plays a video in its entirety without any other command.

Additionally to the BenchLab Client Runtime statistic collection, we collect cpu, memory, disk and network information every second during the experiment on both the client and MediaDrop server. When network bandwidth throttling is needed we use the Linux traffic controller hierarchical token bucket (tc-htb) on the MediaDrop server VM to limit the bandwidth.

3.2 Video metrics analysis

When the browser requests a page from the MediaDrop server, it not only needs to download the main HTML content but also all the referenced resources such as CSS, Javascript and video thumbnails. The details of these Web interactions recorded by the BCR in the HTTP Archive (HAR) format are presented in Figure 5. The 45MB download of the video stream lasts more than 5 minutes and has been truncated from the screenshot. That level of detail is not enough to know if the video has been playing successfully or what the user perceived experience has been. This is why if all HTTP requests are successful, we need video specific instrumentation and metrics to analyze the video streaming experience. In the next sections, we show how the video specific metrics can help experimenter diagnose if a video played smoothly and if not what the root cause of the issue may be.



GET meta-hover-bg.png	200 OK	165 B	6ms
GET e115d410-69a8-4084-abe4-555ff2f3	200 OK	3.1 KB	282ms
GET meta-hover-share.png	200 OK	3.3 KB	31ms
GET appearance.css	200 OK	1.1 KB	10ms
GET mcore-compiled.js	200 OK	23.7 KB	191ms
GET media-view-bg.png	200 OK	124 B	4ms
GET 21s.jpg	200 OK	1.7 KB	177ms
GET comments.css	200 OK	948 B	177ms
GET base.css	200 OK	5.2 KB	44ms
GET duration-left.png	200 OK	331 B	29ms
GET nav-btn-bg.png	200 OK	284 B	86ms
GET 15l.jpg	200 OK	10.9 KB	54ms
GET nav-box.png	200 OK	1.5 KB	10ms
GET nav-btn-edge-outer.png	200 OK	1.2 KB	18ms
GET likes.png	200 OK	167 B	7ms
GET nav-player-edge-outer.png	200 OK	932 B	4ms
GET nav-btn-edge-inner.png	200 OK	787 B	79ms
GET media-view-top.png	200 OK	409 B	32ms
GET search-right.png	200 OK	2.2 KB	5ms
GET search-left.png	200 OK	619 B	43ms
GET comment-bg.png	200 OK	125 B	7ms
GET meta-hover-bottom.png	200 OK	905 B	5ms
GET meta-hover-top.png	200 OK	860 B	6ms
GET nav-player-icons.png	200 OK	1 KB	6ms
GET button-right.png	200 OK	1.5 KB	6ms
GET duration-right.png	200 OK	306 B	9ms
GET footer-bg.png	200 OK	1 KB	43ms
GET comment-top-divot.png	200 OK	631 B	36ms
GET comment-bottom.png	200 OK	425 B	34ms
GET button-left.png	200 OK	763 B	33ms
GET views.png	200 OK	209 B	7ms
GET 14-e115d410-69a8-4084-abe4-555fl	200 OK	45.1 MB	
GET media-view-bottom.png	200 OK	425 B	389ms
GET favicon.ico	404 Not Found	1.8 KB	1.13s
GET jquery-1.11.0.min.js	200 OK	38.1 KB	85ms
35 Requests		45.2 MB	

Figure 5. Typical web interactions between the browser and MediaDrop server when requesting a page and playing a video through an HTTP stream.

3.2.1 *Behavior with unlimited bandwidth*

In our first experimental use-case, we compare the behavior of Firefox, Chrome and Internet Explorer by playing a single SD video of 179 seconds with no bandwidth limitation between the client and the server. The results are shown on Figure 6.

Both Firefox (on the left) and Internet Explorer (on the right) load the entire video in their buffer upfront and play the video. Chrome seems to have a more conservative approach (middle graph) where the buffer stays about 100 seconds ahead of the current video position. The HAR view is however similar for all browsers showing only the overall download time of the video.

When we look at the bandwidth usage at the network level, the 3 browsers download the video right away in a very similar fashion. This means that for Chrome, the video buffer size is independent from the network transfer. Even if more data is available, the video buffer size only expands as the video progresses.

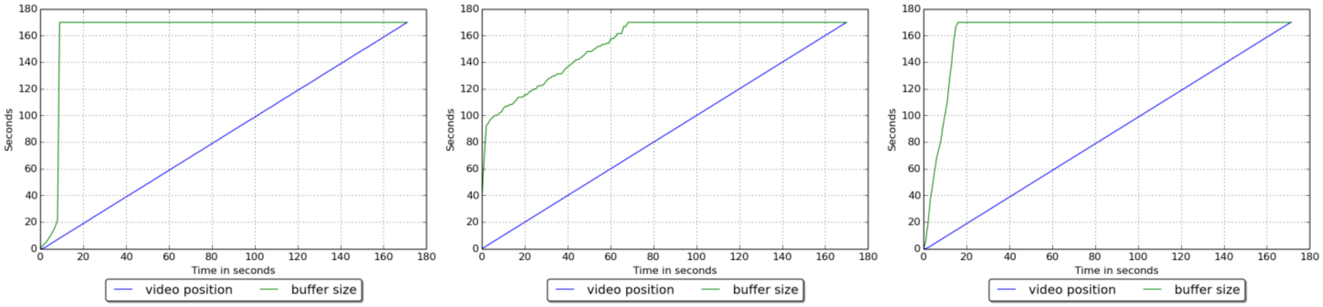


Figure 6. Video buffer behavior for Firefox 30, Chrome 36 and IE 11 (left to right) under normal network conditions

Recall that the buffer size is computed as the difference between buffer end position (B_e) and start position (B_s). In all cases, B_s does not change and only B_e increases until the whole video is buffered. As desktops have enough memory to buffer even our largest video there is no reason for the player to drop parts of the data already played (by moving B_s) as the user might later replay part of or the whole video.

3.2.2 *Detecting replay issues*

Next, we experimented with limiting the server bandwidth to find out what would be the minimum bandwidth a browser would need to play a video of our dataset. We chose an HD video of 6m29s with a bit rate of 965 kbits/sec. We started to limit the video server throughput to 1900kbps which should have been plenty for the browsers to play the video smoothly.

We started with the Chrome browser on an EC2 m1.small instance and obtained the results shown on Figure 7. We notice several drops in the bandwidth usage from the server side as the player stops fetching data when its buffer reaches a certain size. After a very brief pause at the beginning, the video plays without issues until the end with no lag or skips observed.

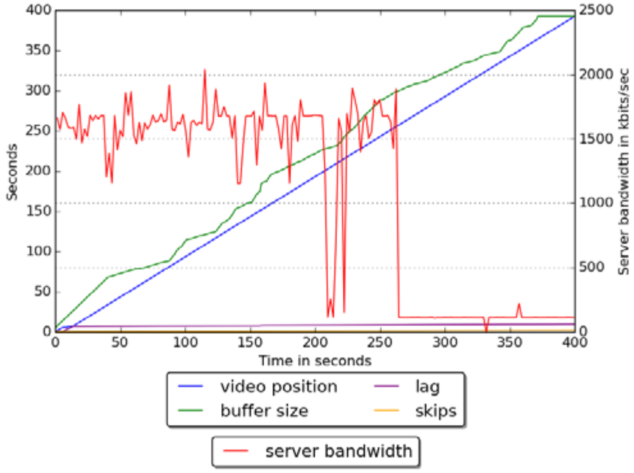


Figure 7. Chrome playing a 965kbps bitrate video with a bandwidth limit of 1900kbps on an EC2 m1.small instance.

We then repeated the same experiment with Firefox in the same m1.small VM and noticed anomalies during replay as shown in Figure 8. First, we notice a number of interruptions in the rendering of the stream highlighted by the *lag* metric on Figure 8. Second, we observe 7 occurrences of parts of the video being skipped highlighted by jumps in the *skips* metric. The accumulated *lag* observed jointly *skips* is a manifestation of a CPU starvation issue. The player cannot play the frames fast enough and skips parts of the video to catchup with the stream. The player skips so many frames that the 400 seconds video ends up being played in 300 seconds. We confirmed by looking at our CPU monitoring data that the CPU was indeed maxed out while the video was playing during network transfers.

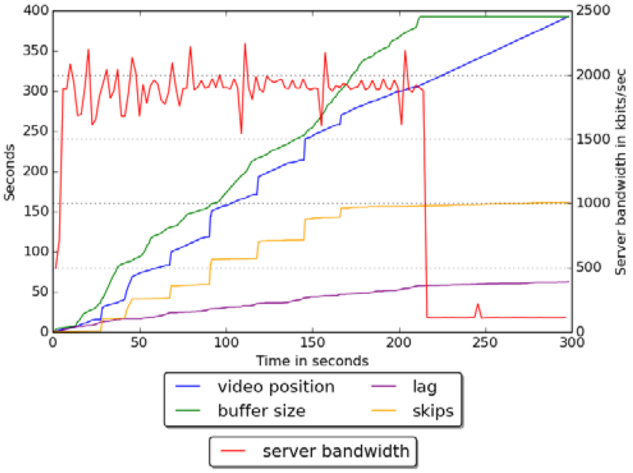


Figure 8. Firefox playing a 965kbps bitrate video with a bandwidth limit of 1900kbps on an EC2 m1.small instance.

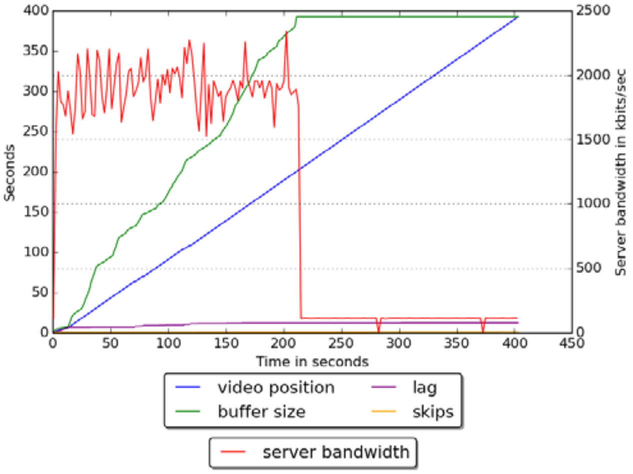


Figure 9. Firefox playing a 965kbps bitrate video with a bandwidth limit of 1900kbps on an EC2 m3.medium instance.

To confirm that CPU was the cause of the observed performance problems, we repeated the experiment using a larger m3.medium EC2 instance under the same network conditions. These instances use a High Frequency Intel Xeon E5-2670 v2 (Ivy Bridge) processor that is about 65% faster than the one found in the cheaper m1.small instances. The results can be seen in Figure 9. The download bandwidth is exactly the same but now the video is playing smoothly with no lags or skips after the startup phase.

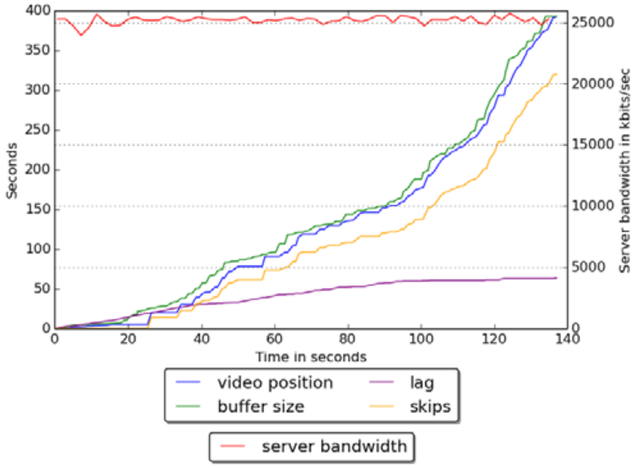


Figure 10. Performance issues on a node deployed in South America during an Internet scale experiment

During one of our Internet scale experiments under high concurrency, we observed the performance shown on Figure 10. The node was competing for network bandwidth with 23 other browsers and had erratic network stream performance. These extreme conditions put the player in a condition that it doesn't seem to be able to cope with as it skips the video much faster than clock time. The 6.5 minute video ends up being replayed in just 140 seconds with most of its content being skipped. Video BenchLab can push the limits of real players in real network conditions which is useful for both video player and network protocol developers.

3.2.3 Detecting network congestions

In this experiment, we re-use the same video as in section 3.2.1 but limit the MediaDrop server bandwidth slightly over the bitrate of the video and induce network congestion by limiting the bandwidth down to 100kbps for a 30 second period during the replay.

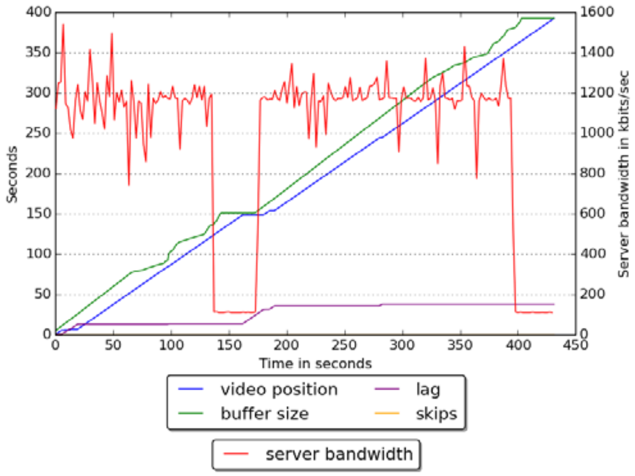


Figure 11. Chrome 36 native HTML5 video player behavior with an induced network congestion.

We first perform the experiment with Chrome and observe the results shown in Figure 11. The video starts to play normally until we reach the network congestion phase. At that point, the buffer size stops growing and the video plays until the current video position reaches the buffer end (B_e). At that point the video stops playing until the download resumes with the restoration of the network bandwidth. The video then plays normally until the end.

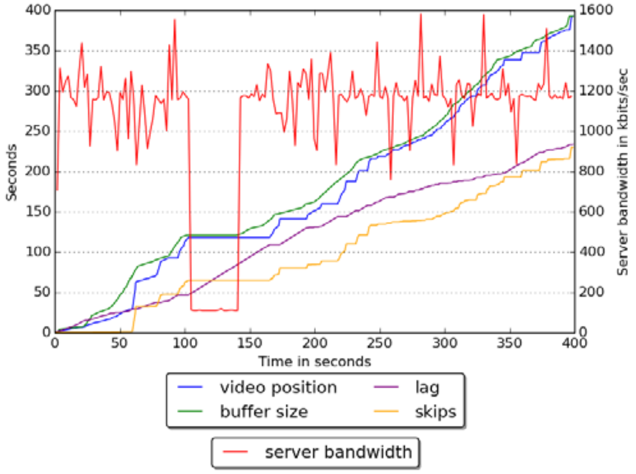


Figure 12. Firefox 30 native HTML5 video player behavior on an m1.small EC2 VM with an induced network congestion.

We repeat the same experiment with Firefox on an m1.small instance and present the results in Figure 12. The CPU limitations are still present during the download phases and characterized by the combination of *lag* and *skips*. However, the network congestion phase is still clearly isolable when the buffer size plateaus and the current video position (V_c) matches the buffer end (B_e). At this point, *lag* increases linearly.

To eliminate the CPU related lag, we repeated the experiment using an m3.medium instance and obtained results similar to the ones observed with Chrome on Figure 11.

3.2.4 Summary

Traditional Web or network level metrics are not enough to determine if a video playback in a browser is satisfactory. We have shown that *lag* and *skips* metrics are good indicators of abnormal video replays. When both *lag* and *skips* are observed simultaneously, it indicates a lack of resources to play the video at the required frame rate. When *lag* increases steadily and the current video position has reached the video buffer end, it indicates a network congestion condition. *skips* could also be observed in case of a corrupted data stream but we would not expect to see any *lag* in that case. These use cases illustrate how Video BenchLab may be used to conduct experiments, analyze results and understand client, server or network behavior.

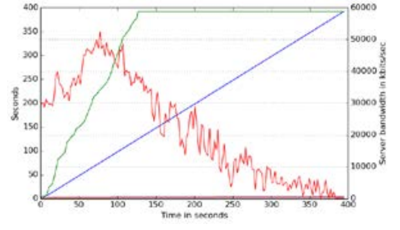
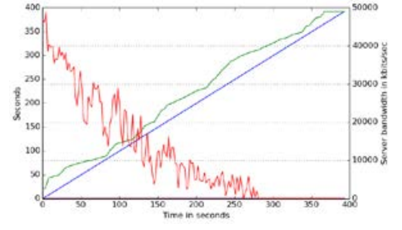
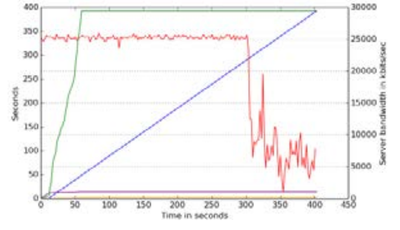
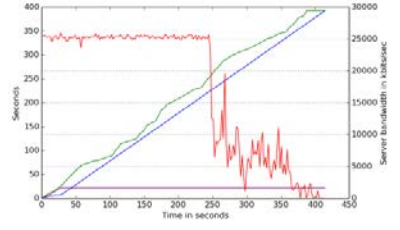
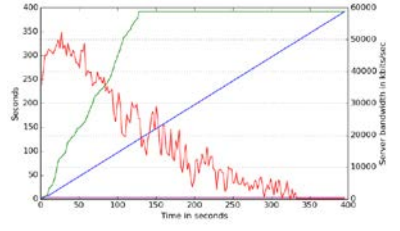
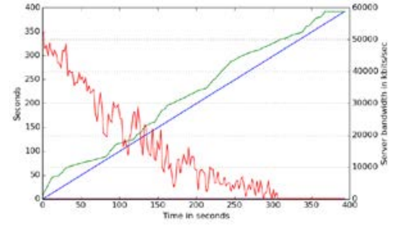
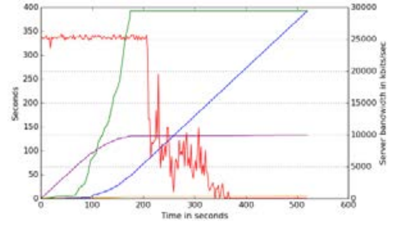
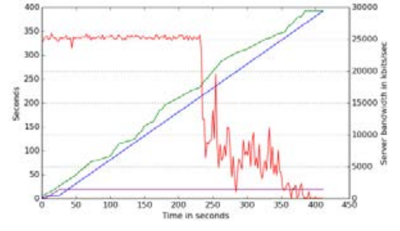
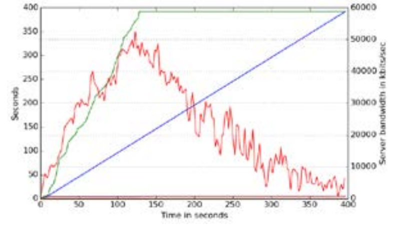
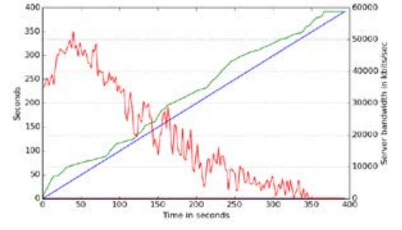
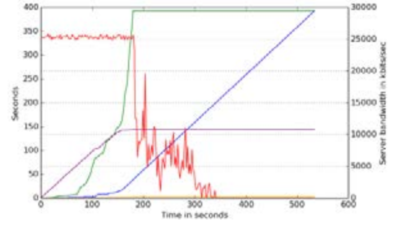
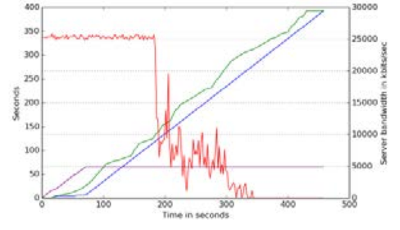
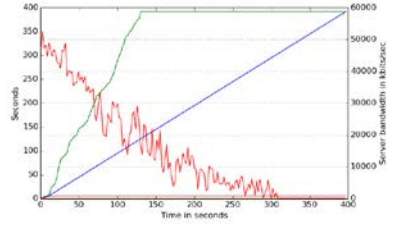
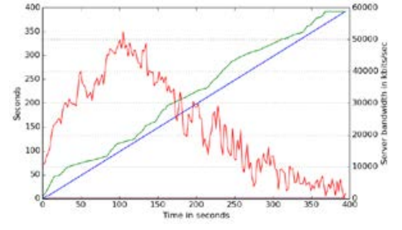
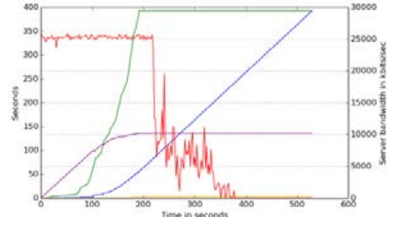
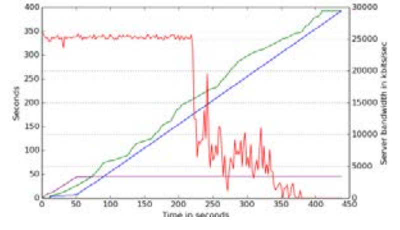
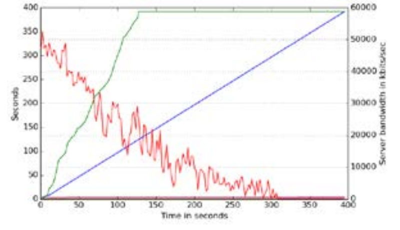
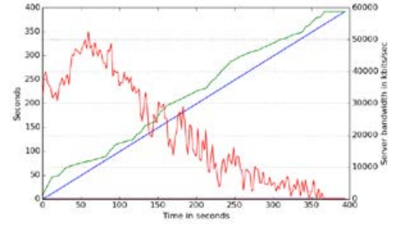
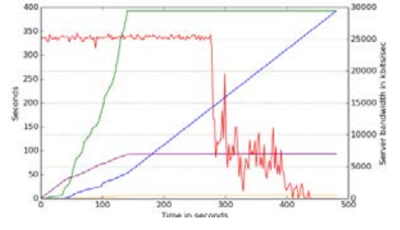
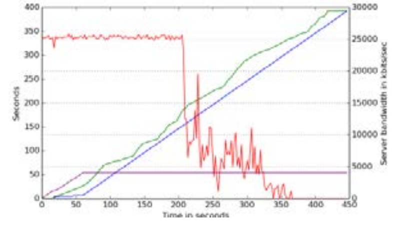
3.3 Internet scale experiments

To illustrate the automation capabilities of Video BenchLab and the ease of Internet scale experimentation, we deployed BCRs with both Chrome (m1.small VM) and Firefox (m3.medium VM) in six Amazon data centers: US East coast, US West coast, Europe, Japan, Australia and Brazil. Our MediaDrop VM is located in the US East coast data center as well as the BenchLab Dashboard as illustrated on Figure 13.

Table 3. Internet scale experimental results

— video position — lag
— buffer size — skips

— server bandwidth

Region	Client side bandwidth limited to 3Mb/s per client		Server side bandwidth limited to 25Mb/s	
	Firefox	Chrome	Firefox	Chrome
US East				
EU				
Japan				
Australia				
Brazil				

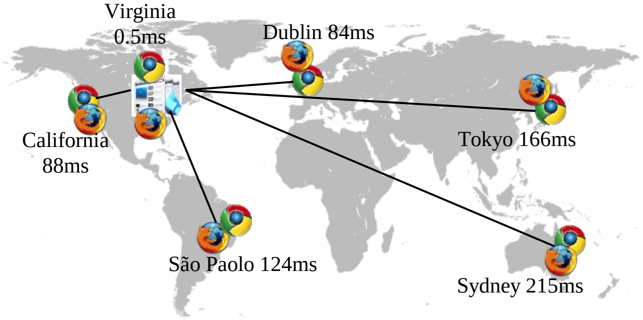


Figure 13. Overview of the Internet scale experiments with 24 browsers deployed in 6 data centers. Average observed latency is to the MediaDrop server deployed in US East.

We conduct 2 experiments: one in which the bandwidth on each client is limited to 3Mb/s to mimic a typical home broadband Internet connection but the server bandwidth is not limited, a second one where clients have unlimited bandwidth but the server is limited to 25Mb/s. We deploy a total of 24 browsers distributed evenly across the 6 data centers. Half of the browsers are Firefox and the other half are Chrome, thus each data center hosts 2 Firefox and 2 Chrome BCRs. The results of both experiments are summarized in Table 3. We only show one browser type per data center per experiment as both browsers behave similarly overall in the same location. Due to space constraints, we do not show the results for the US West coast data center but the results are very similar to the ones observed for the US East coast data center.

All BCRs play the same 6m29s HD video but the start is staggered at a 5 second interval. The video is already in MediaDrop's memory cache before the start of the experiment so that no significant disk IO disturbs the server performance during the experiment.

When the bandwidth is limited on the client side only (first 2 columns of the table), the server bandwidth increases steadily as each new browser starts to fetch the video. The server bandwidth usage peaks at about 50Mb/s when the first browsers are done downloading their video. Note that we use a staggered start every 5 seconds, so that the 24th browser starts 23x5=115 seconds after the first browser. This explains why the server bandwidth curve is shifted on the different client graphs. The bandwidth usage decreases progressively as browsers complete the video download. There is a very minimal lag at the very beginning of the replay that is slightly higher for more remote regions but as soon as the download is started, all browsers are able to play the video without any noticeable lag or skip.

When the bandwidth is limited on the server side only (right most 2 columns of Table 3), the results are dramatically different. As soon as the first browser starts, it uses the entire 25Mbps of server bandwidth. From that point on, all browsers compete for the server bandwidth. All browsers experience lag on startup that increases with the distance from the server. US clients only experience few seconds of lag whereas browsers deployed in Australia or Japan are stalled for more than 100 seconds before the video can start playing. The effect is more pronounced with Firefox that has a more aggressive video buffer management. Chrome experiences at least as much lag as Firefox in the US but seem to do better in remote location with its less greedy approach to video buffer management.

While this experiment illustrates the ability to run large distributed experiments, a researcher would clearly need to run additional experiments to better understand the effects that the video player, network or server can have on the overall performance at Internet scales. Video BenchLab provides the means for researcher to perform these experimentations using real browsers, real networks and realistic workloads and media servers.

4. RELATED WORK

While there has been prior work in the areas of modeling, characterization and evaluation of streaming media systems, there is a relative dearth of benchmarking tools for streaming media researchers. Within the computer architecture community, PARSEC [1] and Versabench [19] are two benchmarking tools that enable performance evaluation of streaming media applications on modern CPUs and multiprocessor architectures. These benchmarking tools are better suited for architecture research and especially for measurements of a single machine, rather than multimedia systems researchers who are evaluating an end-to-end streaming system. Similarly the MediaBench II benchmarking suite [12] includes a comprehensive set of tools for evaluating video encoders and decoders, rather than entire streaming systems. Benchmarking tools and systems for evaluating content-based video retrieval [14] or visual search [15] also exist although they are in a different domain from multimedia systems research. Within the area of multimedia systems, EPFL researchers have developed the Cloud Suite Media streaming benchmark [5]; while this benchmark also uses open source components such as the Darwin streaming server and the Faban workload generator, it is primarily designed for systems that use RTP and RTSP streaming. In contrast, Video BenchLab focuses on HTTP streaming systems, which is the dominant protocol used for streaming content today [19]. Video BenchLab builds upon BenchLab [1] which was designed for web applications and provides similar benchmarking features and benefits for streaming media servers and client-based experimental research.

There has been other related research in the areas of workload characterization and workload generation. G-ISMO [22] is a tool for generating Internet streaming media files and workloads. Techniques for generating HTTP streaming workloads for evaluating server performance have been proposed in [19]; the generated workloads are replayed using the *httperf* tool. Both efforts are complementary to our work in that the generated workload traces can be used by Video BenchLab to inject workloads to a server. Further, unlike replay tools like httperf, our platform uses real web browsers to issue requests and play videos, which yields a more representative HTTP-based streaming workload at the server. Datasets for multimedia benchmarking also exist [12] and numerous such datasets have been released as part of the dataset track within the ACM Multimedia Systems conference [1]. Numerous researchers have characterized the nature of videos found on the Internet. Characteristics of stored videos on the Internet were studied in [19], while live streaming workloads have been studied in [25]. YouTube traffic has been characterized in [20] and [30] while streaming media workloads and Berkeley media workloads have been characterized in [5] and [25]. Companies such as Google regularly measure video quality to different ISPs and publish statistics as a Video Quality report [9]. Finally, in the area of modeling, streaming media servers and their workloads have been modeled in detail in [5] which is complementary to our effort on benchmarking.

5. CONCLUSIONS

Motivated by the need to develop flexible, open performance evaluation tools for streaming media, in this paper we presented the design and implementation of video BenchLab, an open platform tailored for multimedia systems researchers. Components of our platform include a MediaDrop video server, a suite of tools to bulk insert videos and generate streaming media workload, a dataset of freely available video, and a client runtime to replay videos in the native video players of real Web browsers such as Firefox, Chrome and Internet Explorer. Finally, our platform also provides a Dashboard to manage and automate experiments, collect results and perform analytics to compare performance between experiments. We presented numerous illustrative use cases to demonstrate what kinds of experiments could be run using our platform and how the results could be used to better understand system behavior. All the software, datasets, workloads and results used in this paper are made freely available and we hope that the Video BenchLab platform will enable more realistic and scalable experimentation within the research community.

Acknowledgements: This research and the development of BenchLab and Video BenchLab is supported in part by a National Science Foundation grant 1339839.

6. REFERENCES

[1] ACM Multimedia Systems conference Dataset archive, http://traces.cs.umass.edu/index.php/Mmsys/Mmsys

[2] Bienia, Christian, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. "The PARSEC benchmark suite: Characterization and architectural implications." In *Proceedings of the 17th international ACM conference on Parallel architectures and compilation techniques*, pp. 72-81. 2008.

[3] Philip Bräunlich and Gerrit van Aaken – *HTML5 Video Player Comparison* – http://praenanz.de, last update 2014-07-09.

[4] Emmanuel Cecchet, Veena Udayabhanu, Timothy Wood and Prashant Shenoy – *BenchLab: An Open Testbed for Realistic Benchmarking of Web Applications* – Proceedings of 2nd USENIX Conference on Web Application Development (WebApps '11), June 15-16, 2011, Portland, OR.

[5] Cherkasova, Ludmila, and Loren Staley. "Building a Performance Model of Streaming Media Applications in Utility Data Center Environment." In *CCGRID*, vol. 3, p. 52. 2003.

[6] Chesire, Maureen, Alec Wolman, Geoffrey M. Voelker, and Henry M. Levy. "Measurement and Analysis of a Streaming Media Workload." In USITS, vol. 1, pp. 1-1. 2001.

[7] The CloudSuite Media Streaming Benchmark, http://parsa.epfl.ch/cloudsuite/streaming.html, 2012.

[8] Ganglia Monitoring system - http://ganglia.sourceforge.net/.

[9] Google Video Quality Report, https://www.google.com/get/videoqualityreport/

[10] Phillipa Gill, Martin Arlitt, Zongpeng Li, and Anirban Mahanti– *YouTube Traffic Characterization: A View From the Edge* – IMC’07, October 24-26, 2007, San Diego, CA.

[11] HTTP Archive specification (HAR) v1.2 - http://www.softwareishard.com/blog/har-12-spec/.

[12] M. Larson, M. Soleymani, M. Eskevich, P. Serdyukov, R. Ordelman, and G. Jones. "The community and the crowd: Developing large-scale data collections for multimedia benchmarking." IEEE *Multimedia*, (2012).

[13] C. Lee, P. Miodrag, and W. H. Mangione-Smith. "MediaBench: a tool for evaluating and synthesizing multimedia and communications systems." In *Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture*, pp. 330-335. 1997.

[14] C. Leung, and H. Ho-Shing Ip. "Benchmarking for content-based visual information search." In *Advances in Visual Information Systems*, pp. 442-456. Springer 2000.

[15] Libav - Open source audio and video processing tools - http://libav.org/.

[16] S. Marchand-Maillet, and M. Worring. "Benchmarking image and video retrieval: an overview." In *Proceedings of the 8th ACM international workshop on Multimedia information retrieval*, pp. 297-300, 2006.

[17] MediaDrop - http://mediadrop.net/.

[18] Meeyoung Cha, Haewoon Kwak, Pablo Rodriguez, Yong-Yeol Ahn, and Sue Moon – *I Tube, You Tube, Everybody Tubes: Analyzing the World’s Largest User Generated Content Video System* – IMC’07, October 24-26, 2007, San Diego, CA.

[19] Li, Mingzhe, Mark Claypool, Robert Kinicki, and James Nichols. "Characteristics of streaming media stored on the Web." *ACM Transactions on Internet Technology (TOIT)* 5, no. 4: 601-626, 2005.

[20] Gill, Phillipa, Martin Arlitt, Zongpeng Li, and Anirban Mahanti. "Youtube traffic characterization: a view from the edge." In *Proceedings of the 7th ACM Internet Measurement Conference*, pp. 15-28. 2007.

[21] Rabbah, Rodric M., Ian Bratt, Krste Asanovic, and Anant Agarwal. "Versatility and versabench: A new metric and a benchmark suite for flexible architectures." *MIT LCS Technical Report MIT-CSAIL-TR-2004-039*, 2004.

[22] Jin, Shudong, and Azer Bestavros. "Gismo: a generator of internet streaming media objects and workloads." *ACM SIGMETRICS Performance Evaluation Review* 29, no. 3 pages 2-10 2001.

[23] Jim Summers, Tim Brecht, Derek Eager, and Bernard Wong “Methodologies for generating HTTP streaming video workloads to evaluate web server performance”, *Proceedings of the 5th Annual International Systems and Storage Conference (SYSTOR '12)*. New York, NY, 2012.

[24] Selenium - http://seleniumhq.org/.

[25] Slingerland, Nathan T., and Alan Jay Smith. "Design and characterization of the Berkeley multimedia workload." *Multimedia Systems* 8, no. 4: 315-327, 2002.

[26] Standard Performance Evaluation Corporation (SPEC) Benchmarks, www.spec.org

[27] Veloso, Eveline, Virgilio Almeida, Wagner Meira, Azer Bestavros, and Shudong Jin. "A hierarchical characterization of a live streaming media workload." In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pp. 117-130. 2002.

[28] Vimeo Creative Commons / Attribution Licensed video - http://vimeo.com/creativecommons/by.

[29] WebMetrics BrowserMob proxy - http://opensource.webmetrics.com/browsermob-proxy/.

[30] Michael Zink, Kyoungwon Suh, Yu Gu, and Jim Kurose, Characteristics of YouTube Network Traffic at a Campus Network - Measurements, Models, and Implications. *Elsevier Computer Networks.* Vol. 53, No. 4, March 2009.