Spring 2017

CLOUD LIVE VIDEO TRANSFER

Ryan Babcock
Eastern Washington University

CLOUD LIVE VIDEO TRANSFER

A Thesis

Presented To

Eastern Washington University

Cheney, Washington

In Partial Fulfillment of the Requirements

for the Degree

Master of Science in Computer Science

By

Ryan Babcock

Spring 2017

THESIS OF RYAN BABCOCK APPROVED BY

Yun Tian DATE May 31. 2017

NAME OF CHAIR, GRADUATE STUDY COMMITTEE

Carol Taylor DATE May 31, 2017

NAME OF MEMBER, GRADUATE STUDY COMMITTEE

[illegible] DATE May 31, 2017

NAME OF MEMBER, GRADUATE STUDY COMMITTEE

Abstract

As multimedia content continues to grow, considerations for more effective storage options, like cloud technologies, become apparent. While video has become a mainstream media source on the web, live video streaming is growing as a prominent player in the modern marketplace for both businesses and individuals. For instance, a business owner may want to oversee operations while he or she is away, or an individual may want to surveillance their property. In this work, we propose Cloud Live Video Streaming (CLVS) - a very efficient method to stream live video that creates a separate pricing model from modern video streaming services. The key component to CLVS is Amazon Simple Storage Service (S3), which is used to store video segments and metadata. By using S3, CLVS employs what is referred to as a ”serverless” design by removing the need to stream video through an intermediary server. CLVS also removes the need for third party accounts and license agreements. We implement a prototype of CLVS and compare it with an existing commercial video streaming software - Wowza Streaming Engine. As live video streaming becomes more common, alternative and cost effective solutions are essential.

Acknowledgement

A special thank you to Yun Tian. His guidance and knowledge about cloud technology, AWS services, video, and networking helped structure the foundations for this project.

Contents

1 Introduction

There is a growing demand for live video streaming and the current markets offer high costs to provide this service. Companies have to maintain servers and storage, while video content increases both in quality and quantity. By using cloud storage services like Amazon Simple Storage Solutions (S3), video streaming can offer competitive performance and higher savings compared to services currently available. Cloud live video streaming (CLVS) uses S3 to eliminate the need for server rental or ownership. This system is highly applicable to the current demand in streaming technology.

The streaming video industry has become a fast growing cable competitor. The growth of the streaming video industry has risen from \$3.1 billion in 2013 to \$4 billion in 2014 to \$5.1 billion in 2015, and predicted to be \$6.7 billion in 2016 [1]. There are also many households who do not have a traditional TV subscription and instead rely on streaming services for video. Analysts found that 20.4% of households fell into this category in 2015, up from 18.8% in 2014. By the end of 2016, this number is predicted to be 21.9% [1]. With the streaming video industry on the rise, live video streaming is a requirement for such events as: the Olympics, the Super Bowl, emergency broadcasts, concerts, presidential debates, etc.

A large number of businesses have recently explored live streaming. According to a recent survey [2], in the past 12 months 44% of businesses have conducted one or more live streaming video events, 35% do not plan to test or conduct a live streaming event, and 20% plan to test one or more live streaming events. The large participation by the business community can be attributed to the growing availability of smart devices along with the integration of widely-used social media platforms. As video streaming becomes more available, the opportunities to reach potential clients do as well.

In 2015, Facebook launched a new live streaming feature for celebrities, which slowly became available to its users. In 2016, Facebook announced that their searching algorithm will now prioritize live streaming videos. Also in 2015, YouTube re-launched its live streaming feature with a focus on e-sports and gaming. Live streaming features are now being offered by large renowned companies, and their recent involvement in social media and video sharing suggests a growing demand for live streaming services. This creates more opportunity for widespread personal use and event sharing.

It is clear that there is rapid growth in the streaming video industry, and live video is being implemented by some of the largest video streaming companies in the world. Most companies that offer live video streaming services charge by bandwidth used, customer access, storage costs, license agreements, and viewer hours [3] [4] [5]. The cost of using CLVS uses a similar pricing model, but alleviates license agreements. In many cases, the price of CLVS is much lower than using modern live video streaming services, while offering very comparable performance. The prospects CLVS are very promising in today's market.

CLVS uses some fundamental video streaming techniques, but has some key differences. The primary development with CLVS is its *serverless* architecture. Instead of using intermediary servers, CLVS uses S3 as a shared space to transfer data. In order to access the stream, the viewer uses the appropriate bucket name and key, along with valid credentials, and the most recent video segment is displayed. There are many benefits to this technique, including scalability, cost, access to additional cloud resources like content delivery networks (CDNs), and security. This elegant design simplifies modern video streaming practices, removes the need for dedicated servers, and provides a great alternative to live video streaming.

The paper is organized in the following manner. Section 2 examines past research that has led to CLVS. Section 3 explains the methodology carried out in implementing CLVS. Section 4 describes how CLVS and other modern live video streaming techniques were tested and compared. Section 5 discusses the results found in section 4 and explores appropriate applications for CLVS. Section 6 presents the necessary improvements made apparent from the examination in Section 5 and the future direction of the project. The paper ends with concluding remarks in Section 7.

2 Related Works

There are many protocols associated with video streaming. CLVS does not make an attempt to follow any standards developed by the International Telecommunications Union (ITU) or the Internet Engineering Task Force (IETF). Rather, video streaming protocols are considered to provide a model or framework for CLVS functionality. In addition, for a more in-depth comparison, it is beneficial to show the current transport protocols that are popular among modern video streaming practices.

2.1 Modern Video Streaming

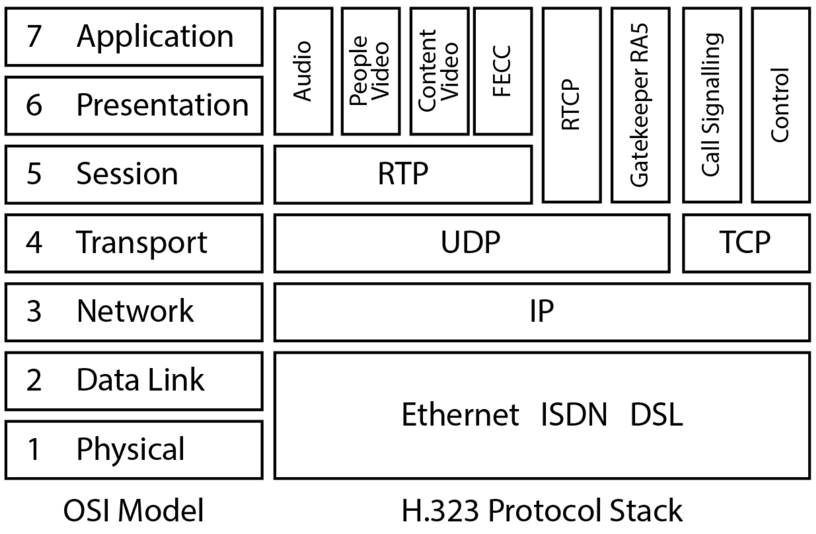


Figure 2.1: Video Streaming and Open Systems Interconnection Model [7]

The potential ways to stream video are vast. Some of these different ways include progressive streaming, true streaming, adaptive streaming, on-demand streaming, and real-time streaming [8]. There are also more transport specific protocols such as the Real Time Transport Protocol (RTP) and the Real Time Messaging Protocol (RTMP). More so, there are many different types of adaptive streaming technologies such as HTTP Live Streaming, Adobe HTTP Dynamic Streaming, Microsoft Smooth Streaming, Dynamic Adaptive Streaming over HTTP (MPEG-DASH), Shoutcast, and BitTorrent Live Streaming. Before exploring these methodologies, its important to see how video streaming uses the protocol stack.

In reference to the Open Systems Interconnection (OSI) model, video streaming takes advantage of the transport layer, session layer, presentation layer, and application layer [8]. Figure 2.1 shows how H.323, a video conferencing protocol, is organized in terms of the OSI model. This provides an example of how live streaming technologies work. With regards to the transport layer, Video-on-Demand (VOD) tends to support the TCP transport protocol for reliability and quality, where as real-time or live streaming generally prefer to use UDP for speed. The session layer is used to organize stream activity into ongoing units such as movies and broadcasts. The presentation layer manages the bridge between information seen by the application and information as sent over the network. And the application layer is used to interpret and display data retrieved from the network.

These layers are all utilized in video streaming architecture.

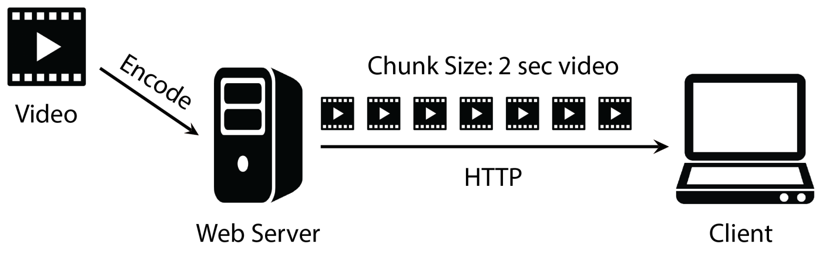


Figure 2.2: Video streaming basics [9]

There are some basic video streaming practices shared amongst many video streaming protocols. Figure 2.2 shows the overall concept of the modern video streaming strategy. First, the video is encoded. To encode a video means to compress a video file and transmit chunks of that file through a bitstream. A bitstream can be viewed as a container such as a MP4, FLV, WebM, ASF, or ISMA. Containers hold the actual video data while providing metadata like title, total length, and index positions (i.e. scene 2). The bitstream can be delivered to a client from a streaming server using RTMP or RTP. RTMP is a TCP-based protocol that splits streams into fragments and uses multiple channels for handling different data streams. For instance, one channel is used for video, another for audio, and another for Remote Procedure Call (RPC) requests. RTMP data is encapsulated and exchanged via HTTP [10]. RTP provides end-to-end delivery for real-time data in unicast and multicast sessions. It also provides timestamping, packet identification, sequence numbering, type identification, and loss detection. RTP also works in conjunction with the Real-Time Control Protocol (RTCP). RTCP monitors statistics like packets sent, number of packets lost, interarrival jitter, etc. This feedback can be used for adaptive media encoding or detection of transmission faults [11] [12].

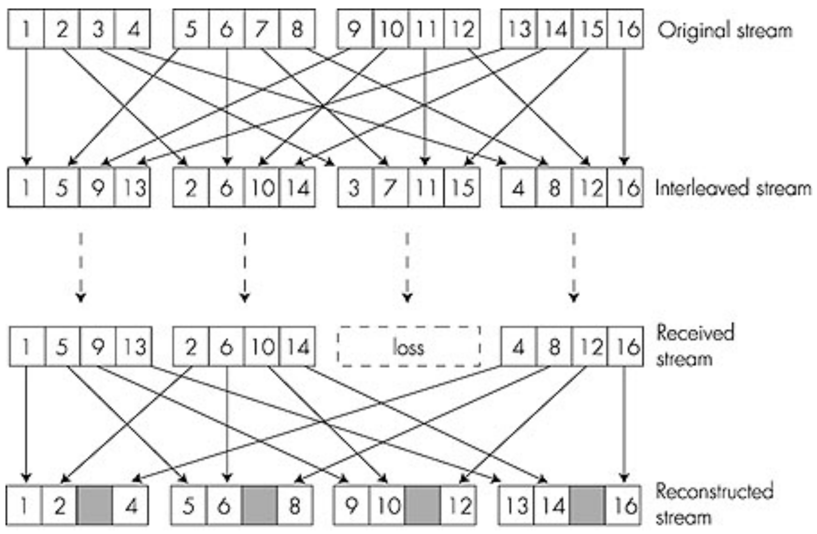


Figure 2.3: FEC interleaving [12]

Video streaming data can be lost or corrupted due to network conditions resulting in packet loss, which can be handled in many different ways. One solution is to use Forward Error Correction (FEC). This technique sends an extra packet of data with a group of 4 packets (5 total). If one packet is missing when received by the client, the other packets can make up that packet with an XOR operation. The disadvantage of this technique is that an extra packet is sent for every transaction. Another solution is interleaving. This technique distributes data among packets so that the video data is not sent sequentially (see Figure 2.3). This causes jitter from packet loss to be spread over a larger chunk of data rather than occurring all at once. Another well-known method to prevent jitter is by filling a large buffer with many data packets before the playback of a video. This method costs more lag before playback, but eliminates a certain amount of jitter while viewing the video [12]. Limiting the amount of jitter or lag is one aspect of ensuring Quality of Service (QoS).

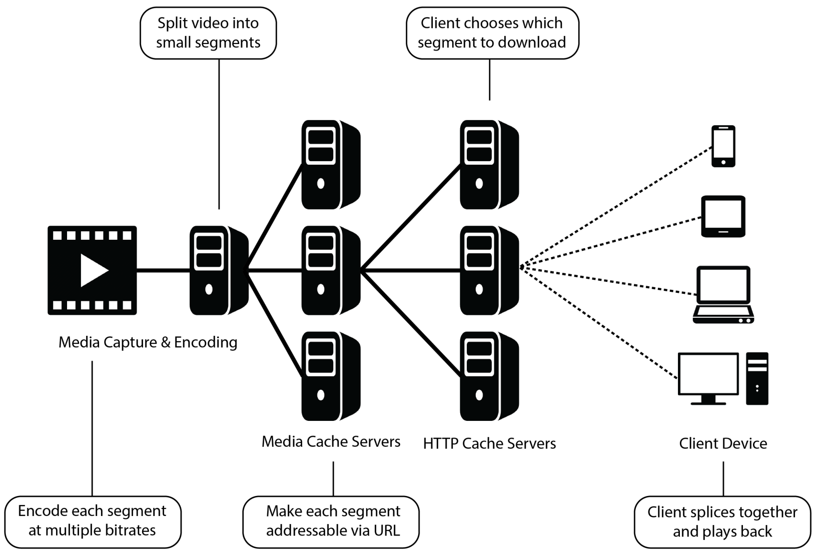

Split video into small segments
Client chooses which segment to download
Media Capture & Encoding
Media Cache Servers
HTTP Cache Servers
Client Device
Encode each segment at multiple bitrates
Make each segment addressable via URL
Client splices together and plays back

Figure 2.4: HTTP Adaptive Streaming [13]

Modern video streaming techniques provide many enhancements to the previously discussed principles. Progressive streaming only sends every other pixel row in the beginning to limit the upfront cost of loading the video. This reduces the initial delay of a video, but makes the upfront quality a little worse. True streaming is more specific to the transfer of a video and holds some adaptive capabilities to enhance performance, also known as adaptive streaming. It splits a file into multiple files, adapt streams to the changes in throughput, and operates without direct client interaction. Adaptive streaming can operate with or without a streaming server. The adaptive streaming server implementation requires constant communication between the client and server. Adaptive streaming without a streaming server uses different addresses for the different stream qualities and the client shifts from one to the other based on their preferred viewing experience [14] (see Figure 2.4). Adaptive streaming is one of the most common streaming methods for delivering video content today.

Adaptive streaming takes advantage of transcoding, transrating, trans-sizing, and transmuxing. *Transcoding* takes encoded content, decompresses (decodes) it, alters it, and encodes it again [15]. With transcoding, the video can be altered in different ways. The video or audio could be compressed using a different compression algorithm, and watermarks, logos, or other graphics could also be added. This concept differs from transrating, which specifically makes changes to bitrates. This creates renditions of a video stream by converting a video into one or more lower bitrate streams. *Trans-sizing* alternatively resizes the video frame from higher resolutions to lower resolutions. Finally, *transmuxing* can change the container format to support different platforms. Besides transmuxing, the rest of the operations are the responsibilities of a *transcoder*. Therefore, the term transcoding is sometimes used to describe the tasks of a transcoder. The transcoder is generally a server that may or may not be the same media server that handles transmuxing.

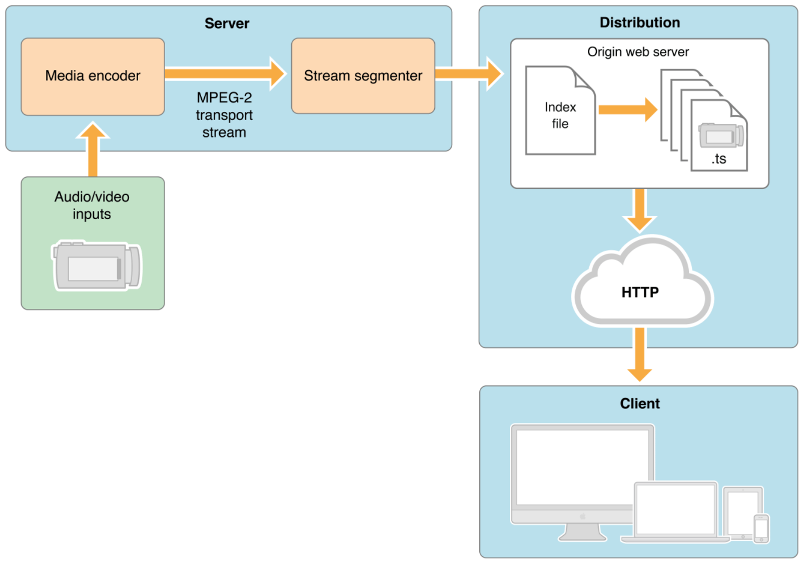


Figure 2.5: HTTP Live Streaming model [6]

HTTP Live Streaming (HLS) is an adaptive streaming protocol developed by Apple (see Figure 2.5)[6]. It follows a lot of the same principles already mentioned: video is encoded into a bitstream, the bitstream is split into chunks, the chunks are split into multiple sized files, a client downloads files based on their available bandwidth, a buffer is filled with chunks of data, and the user views the content of each video chunk within the buffer. HLS uses a transcoder and creates multiple MPEG transport stream (TS) files, which represent the video chunks. TS is a digital container format and helps maintain transmission integrity when the network is degraded. Before a client downloads video content, the appropriate TS file is chosen from a playlist based on their device and available bandwidth.

The design of CLVS is similar to HLS, but also very different. Both CLVS and HLS use encoding, segmentation, and the HTTP protocol. However, CLVS does not use elements specific to a transcoder or any form of adaptive video streaming. Also, CLVS is serverless. All the encoding happens at the video source and segments are uploaded to cloud storage. The client can then download the segments and view them in a sequence.

2.2 Cloud Storage Service

Cloud storage is a scalable way to store, access, and share data over the Internet. There are many different cloud service providers including: Microsoft, Google, Amazon, IBM, Oracle, and Verizon. One of the ways cloud storage technologies provide scalability, reliability, and availability is through replication. Three standard approaches to replicating data are synchronous replication, asynchronous replication, and quorum-based replication.

Synchronous replication stores user data in both a primary location and its replicas. This is ideal to prevent data loss in the event of a failure of the primary node. This process can also scale read capacity for queries that require up-to-date data. The drawback to synchronous replication is that the primary node is interconnected to the replicas, and a transaction cannot be completed until all replicas have written the data. This can cause poor performance and availability when the network is unreliable.

Asynchronous replication removes the adhesion between the primary node and its replicas, but introduces replication lag. This technique is typically used to horizontally scale the system's read capacity for queries that do not require time sensitive data. Asynchronous replication can be used to separate data among data centers, providing an added layer of durability.

Amazon Web Services (AWS) provides a quorum-based replication solution that combines synchronous and asynchronous replication [16]. The primary node is still coupled to its replicas, but does not require a write confirmation. Rather, Dynamo (an AWS service) provides an ”always writable” data store [17]. The gives flexibility to specify write confirmations in specific instances or to always allow the last write to take priority. S3 also replicates data across Availability Zones or data centers to provide fault tolerance [16].

There are many benefits to using cloud storage services. First, the client does not need to purchase and update their own server. It is the responsibility of the cloud service provider to maintain and update their equipment, which can alleviate costs and headaches. Second, the cost of service is relative to the amount of data the client chooses to store and the frequency that data is accessed. This is similar to a utility bill in that clients only pay for what they use. Third, the data is very secure. A client can specify permissions, passwords, and can only modify the data based on credentials provided by Amazon. Finally, the data is highly partition tolerant. Through replication within and across data centers, it is very rare for data to be unrecoverable. In fact, S3 Standard is designed to provide 99.999999999% durability of objects within a given year [18]. The numerous benefits to using cloud storage services makes it a preferable way to store data.

Many cloud service providers maintain an application program interface (API) that allows developers to use their resources. AWS offers many cloud service APIs through their software development kit (SDK). For S3, a developer can manipulate files in the cloud, such upload, download, or delete. In addition, a developer using the S3 API can edit permissions for files and alter certain network behaviors to meet their needs. CLVS uses the AWS SDK to transfer data from the video source to the client.

2.3 Video Compression

Video compression is used to reduce the size of video data, which allows for lower storage costs and lower transmission bandwidth requirements [19]. This is achieved by reducing the information within the video file, which generally reduces the quality of the video itself. Higher levels of compression (also called profiles) result in lower storage cost, but worse video quality. Lower levels of compression result in better video quality, but higher storage cost.

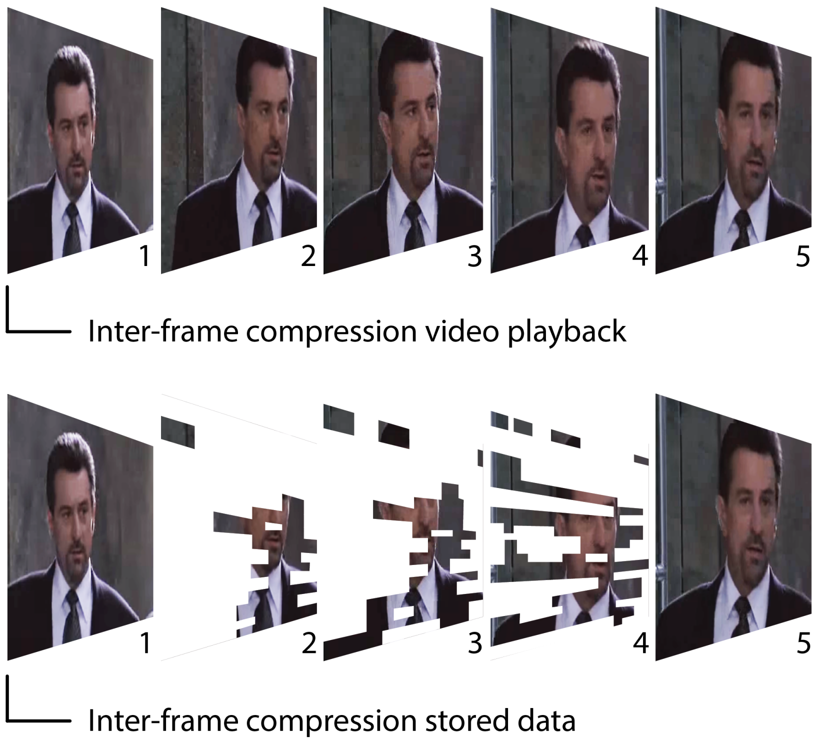

1
2
3
4
5
Inter-frame compression video playback
1
2
3
4
5
Inter-frame compression stored data

Figure 2.6: Inter-frame compression [20]

There are two main types of video compression: intra-frame and inter-frame. Intra-frame compression uses the current video frame as its only reference. In other words, it uses image compression on each frame separately. Inter-frame compression uses preceding and/or succeeding frames in a sequence to compress the contents of the current frame. Figure 2.6 illustrates a common example of inter-frame compression. The entire information of the first frame is captured and the subsequent frame's data is based on any changes from the previous frame. This way, elements that do not change from frame to frame can be ignored.

Better video compression techniques come at the cost of higher CPU utilization and compression latency [19][21]. More efficient techniques (such as H.264) require more arithmetic operations than more simplified techniques, like M-JPEG. Therefore, better compression comes at the cost of higher CPU utilization. To avoid this cost, hardware specific codecs can be used to compress video into a specific format. However, this limits the flexibility of video compression. For instance, a CPU can concurrently compress the same video into multiple formats, bit rates, and resolutions. This type of flexibility may be necessary depending on the video streaming application, whereas hardware based solutions use fixed formats. Along with CPU utilization, video compression comes at the cost of time or compression latency. More advanced compression algorithms result in higher compression latency given the same processing power [21]. More efficient compression techniques are still preferred in modern video streaming practices despite these drawbacks.

2.4 Video Streaming with Cloud Services

There has been some similar research with regards to live video streaming using a cloud infrastructure. Karakasilis et al. [22] streams live video using GRID and cloud infrastructures, but use their own servers as distributors for their video stream. The goal of their research was to demonstrate that live video streaming was possible using distributed computing, but do not focus on using existing cloud services. Li et al. [23] focus on transcoding a live video stream based on QoS demands using virtual machines. The virtual machines they used was based on Amazon Elastic Compute Clound (EC2) machines (t2.micro) that can be utilized at very low costs. Though their research seems promising, their cost analysis is not well explained. Also, there are no video metrics or throughput measurements to confirm their evaluation.

There was one individual, Carson McDonald, whom experimented with a similar idea to CLVS using an iPhone, HLS, S3, and Amazon CloudFront [24]. In this proof of concept, McDonald dumps short clips of a video stream out to a file using FFMpeg. Each clip and index file is then uploaded S3 using Ruby and Rightscale AWS gem. The client then accesses the index file to locate the correct video segment and views it through a CloudFront URL. McDonald's research realized the possibility and potential of video streaming through a cloud storage application. Our work is the first proposed that comprehensively evaluates live video transfer with cloud storage, in terms of performance, financial cost, and feasibility.

2.5 Commercial Products

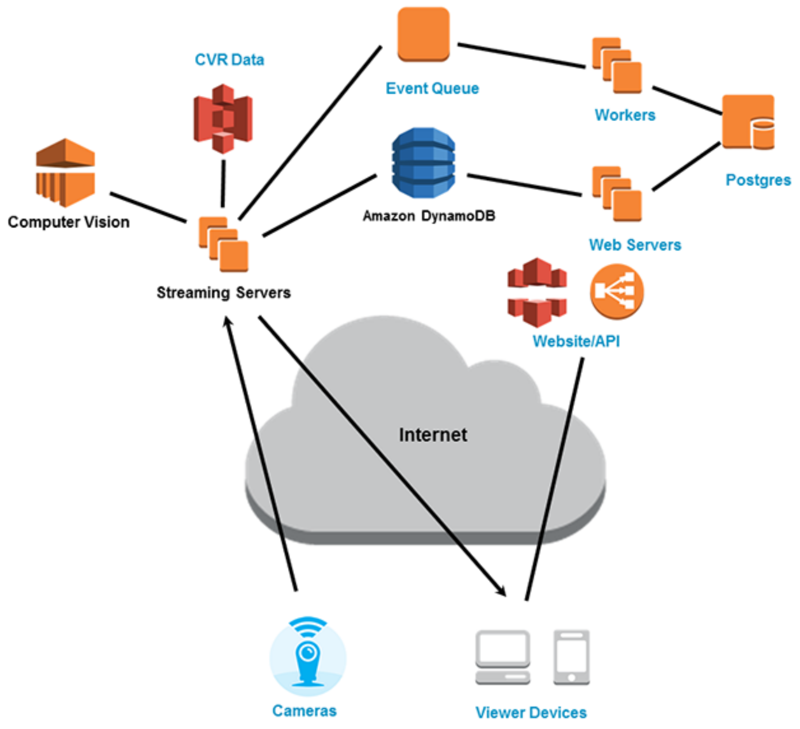


Figure 2.7: Dropcam architecture on AWS [25]

Modern video streaming and cloud camera companies implement and offer their services in different ways. The common approach involves the purchase of a streaming service subscription or cloud camera. Live streaming and cloud camera services also require clients to create accounts in order to access their video feed [26] [27]. Many video streaming service providers will use cloud storage as a way to backup client data [28] [29], however, this service is either limited or requires additional fees. Other companies, in fact, use cloud services with their product, but continue to use servers. Dropcam [25] uses a EC2 video streaming server with S3, while using Amazon DynamoDB to scale and maintain throughput (see Figure 2.7). Netflix uses AWS exclusively and runs up to 100,000 server instances [30]. Though Netflix does not currently support live video streaming, it is important to note that their software relies heavily on the use of servers.

There are some drawbacks to modern commercial products. First, the upfront cost of a cloud camera is either expensive or very low quality. Subscription based streaming is also expensive, and lower cost subscriptions generally contain advertising and limited services. Second, the account requirement means that client data is not private. The video stream provider has the ability to view specific client data. It is not expected that a streaming provider will access client video content directly, but the possibility is there. Finally, video stream providers may release data to second and third parties. Currently, data mining through visual characteristics is in its infancy. However, higher use of live video streaming creates a platform for data gathering. Google's Nest Cam [31] is a cloud camera that uses advanced analysis to learn more about what is happening in the video. The intent is to notify customers of alarming conditions, and Nest states that privacy is protected. However, sending sensitive video content through a company's server relies on the trust of the consumer. The CLVS prototype shows some potential applications that can avoid many of these issues through cost, efficiency, and private access.

3 Our Proposed Methodology

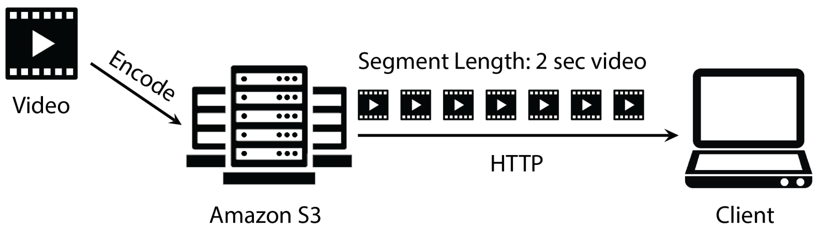


Figure 3.1: CLVS basics

The implementation of CLVS follows similar principles discussed in Section 2.1. A video source records an event, every frame is compressed, a chunk of compressed frames are sent to cloud storage, a client pulls each chunk from cloud storage and views the video. Figure 3.1 illustrates the similarity between the CLVS model and modern video streaming (see Figure 2.2), while Figure 3.2 shows the key operations within CLVS. This solution also differs from some common practices. The goal of this project is to assess the practicality and efficiency of CLVS when compared to modern video streaming practices. CLVS uses a very efficient design by employing parallel programming techniques, prefetching, and a producer consumer model.

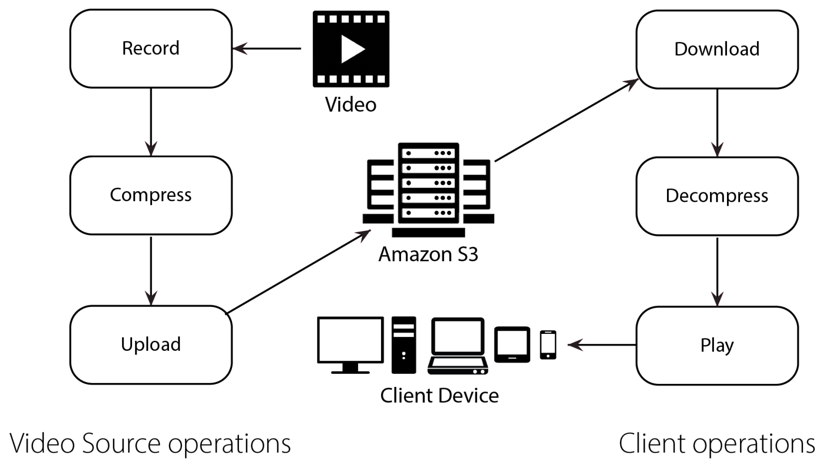


Figure 3.2: CLVS overall concept

3.1 Video Source

The recording device used to capture and send the video data is referred to as the *video source*. CLVS uses a Raspberry Pi 3 with a 5MP camera module, which provides moderately powerful computing power, high networking capability, high customization options, and cost effectiveness. Though capable of higher specifications, the recorded video uses minimal qualifications - no color, no sound, and low frames per second (FPS). Maintaining a primitive video quality allows for simple implementation and testing. The video source has many responsibilities, including: preparation, compression, segmentation, transportation of segments to S3, removal of old segments from S3, and recording performance data.

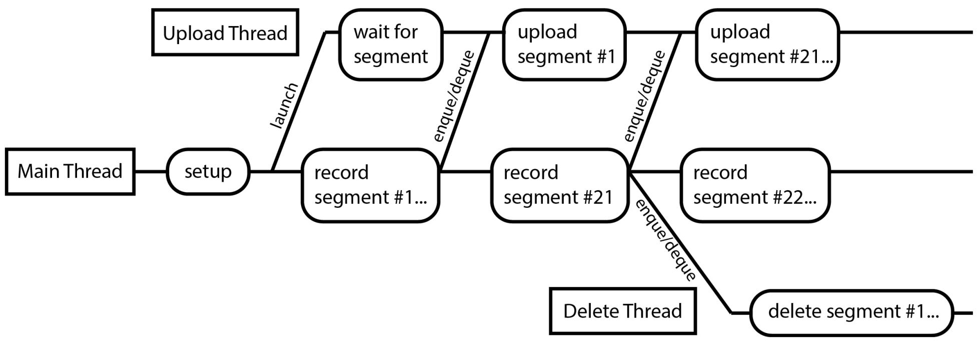


Figure 3.3: Video source thread model

The video source uses three main threads to perform all the above operations, namely the main thread, upload thread, and delete thread (see Figure 3.3). The main thread initializes the setup parameters, such as FPS, segment length, and start time. The other two threads are then launched with shared resources. After setup, the main thread continues to record video, transfer segments to the upload thread, and signal the delete thread. The upload thread is used to upload all recorded segments to S3. The upload thread maintains a direct connection with the S3 bucket, and communicates with the delete thread to send delete requests to S3. Finally, the delete thread is used to delete older segments based on the max segments parameter. When the S3 bucket contains max segments, the delete thread removes the oldest segment when the newest one is uploaded. All threads communicate through multiple queues.

3.2 Compression

An efficient way to compress video is through inter-frame compression. The drawback, however, to using open source libraries is the availability of different compression algorithms. Some Java libraries did present this option, but required that the compressed video segment be written to hard disk. It is more efficient, though, to store the compressed segment into main memory and then directly transfer it to S3. Also, every video segment would have to be written to disk before it could be read from the video player. One library provided h.264 compression, but stored data inefficiently and resulted in a flawed algorithm with 4x the size output. Therefore, CLVS does not utilized advanced inter-frame compression.

CLVS uses intra-frame compression to avoid multiple IO operations. A built-in Java compression library is utilized where every frame is compressed using JPEG encoding. Though not as efficient as inter-frame compression, JPEG encoding still provides a 11.5:1 average data reduction. By using intra-frame compression, the overhead and latency of file IO is avoided. By focusing on bit rate, rather than image quality, the potential of CLVS can be evaluated without using an optimal compression algorithm.

3.3 Segmentation

Every recorded video segment is based on the setup parameters mentioned in Section 3.1: start time, segment length, and FPS. Start time is the timestamp of when the video source began recording. Segment length is the number of seconds that each segment will play. FPS is the number of frames that will appear within a second of viewing. Therefore, each segment will contain $segmentlength * FPS$ frames.

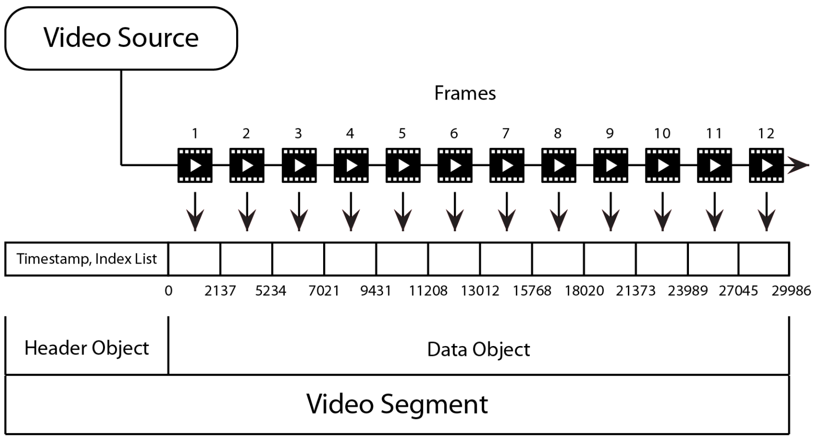


Figure 3.4: Video segment structure

For each video segment there is a header object and data object as shown in Figure 3.4. The data object contains a buffer that holds all the recorded compressed frames. The header object contains both a buffer of indeces, called the *index list*, and a timestamp. The index list contains the index positions of each frame within the data object. The timestamp is used to to get the current length of the recorded event. It is recorded before the first frame is saved to the data buffer, and subtracts the start time.

Maximum video index and maximum segments are used to help define segment metadata. When a video segment is recorded it is given a name based on an index (ie. myvideo0, myvideo1, myvideo2, etc). Maximum video index determines the largest index limit, where upon reaching it the current index is reset to zero. Maximum segments determines the limit of segments that can be stored within S3 at one time. This controls the space occupied by video content within S3.

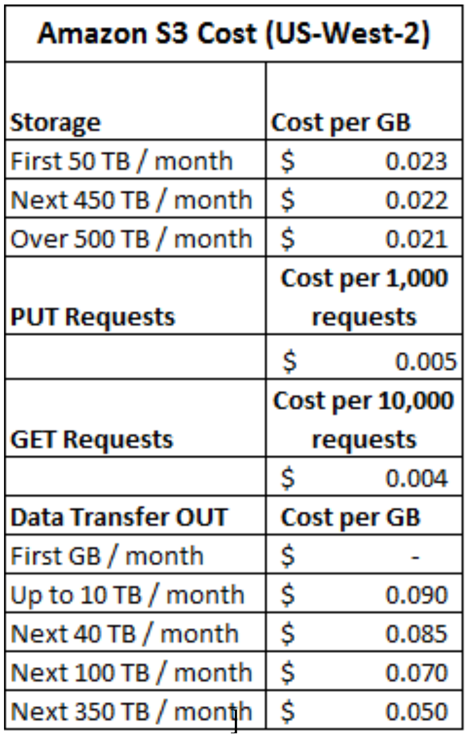
3.4 Cloud Storage

Cloud storage, or S3 in this case, is used to store all recorded video segments. In order to use S3 the user, both sender and receiver, must provide accurate credentials. This is a file provided by Amazon that contains both a username and an security key. Before a bucket can be accessed, the user and key must be verified by Amazon. This adds a layer of security to prevent unwanted access to user files.

Any file stored in S3 is referred to as an *object* and the object name is called a *key*, not to be confused with the security key. Before a segment can be uploaded to an S3 bucket it must provide a key. In this case, the key is the segment name mentioned in Section 3.3. At the end of all recordings, all segments are removed from S3, although could be kept for further processing. The only other file that is put into the bucket is the setup file, which is done before any segments are uploaded. The number of segments that can be in the bucket at one time are based on the maximum segments parameter mentioned in Section 3.3.

CLVS uses the TransferManager class in the S3 API, which manages multiple threads and connections to expedite the transfer of large files when high bandwidth is available. Underlying, S3 uses HTTP, TCP, and TLS 1.2 to transfer data. HTTP and TCP are designed for reliable transfer of files. Furthermore, these protocols are recently being used for video streaming [8]. Though slower than UDP, TCP has the benefit of congestion control as well as better pre-fetching and buffering capabilities [32]. Many streaming platforms are configured to support web services, including content delivery networks (CDNs). Therefore, HTTP is commonly used with video delivery. Another benefit with HTTP and TCP is that they are commonly used on the web and are therefore not blocked by firewalls. The TLS protocol is used to encrypt the data and prevent access from malicious users.

The cost of using S3 is based on four primary qualifications: the amount of space used, the number of uploads (PUT requests), the number of downloads (GET requests), and the amount of data that goes out to the Internet (data transfer OUT) as seen in Figure 3.5. The total amount of space occupied can be calculated as $framesize * FPS * segmentlength * maxsegments + setupfilesize$. In the case where framesize = 500kb, FPS = 25, segment length = 4, max segments = 10, and the setup file = 1kb, the total storage space required is 500MB. This means that for a high quality live video stream, the minimum storage space for S3 can be purchased at 1 GB. Therefore, storage cost will remain relatively low. The number of uploads and downloads becomes another concern. PUT requests through CLVS is constant and deterministic based on the segment length param-



Amazon S3 Cost (US-West-2)	
Storage	**Cost per GB**
First 50 TB / month	$ 0.023
Next 450 TB / month	$ 0.022
Over 500 TB / month	$ 0.021
PUT Requests	**Cost per 1,000 requests**
	$ 0.005
GET Requests	**Cost per 10,000 requests**
	$ 0.004
Data Transfer OUT	**Cost per GB**
First GB / month	$ -
Up to 10 TB / month	$ 0.090
Next 40 TB / month	$ 0.085
Next 100 TB / month	$ 0.070
Next 350 TB / month	$ 0.050

Figure 3.5: S3 pricing model [33]

eter. GET requests, in contrast, is very dynamic based on the number of viewers and how long the video is viewed. Likewise, data transfer OUT is based on viewers. Beyond viewers, GET request cost is based on segment length whereas data transfer OUT cost is based on bitrate.

3.5 Client

The client pulls the segments down from S3 and views the video stream. Three threads are used in this process: the main thread, the downloader thread, and the displayer (see Figure 3.6). The main thread initializes the other two threads and communicates with the downloader through two queues. The signal queue is used to transfer setup information from the downloader to the main thread, while the segment queue is used to send video segments from the downloader to the main thread. The main thread uses the information in the setup file to know how many frames are contained in a video segment and the start time of the recording. The downloader uses the maximum video index and maximum segments parameters to know which video index will be most current.

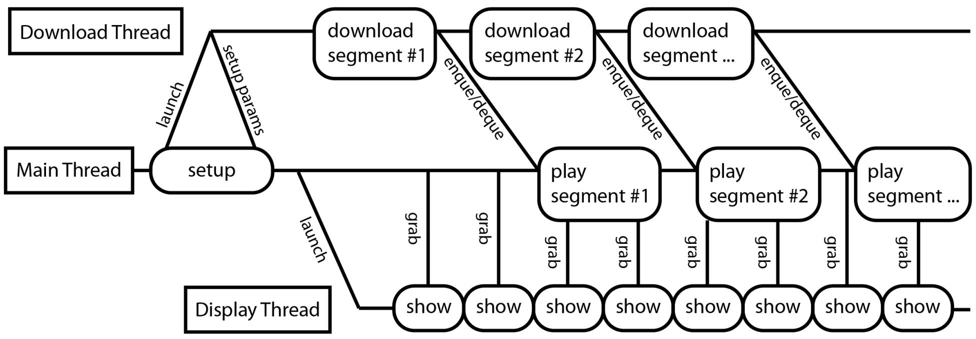


Figure 3.6: Client thread model

The process of viewing the video is quite simple. The transfer of video segments from the downloader to the main thread is a producer-consumer design. The downloader thread downloads the setup file and sends that information to the main thread through the signal queue. Afterwards, the downloader finds the most current video and puts it into the segment queue. The main thread then grabs the video segment from the segment queue, and uses the header within each segment to find each frame. The displayer grabs the frame and displays it in a window. The main thread waits for $1/FPS$ seconds and grabs the next frame. Prefetching allows the downloader to grab the next video segment, while the main thread finishes cycling through the previous one. If there exist more than one segment in the segment queue, the older segments are discarded to reduce delay and allow the viewer to receive the newest segment. This process continues until the user exits the program.

3.6 Considerations

The are no network ”controllers” for CLVS as with the Real Time Streaming Protocol (RTSP). RTSP sends signals back and forth between a client and a server. This allows for operations like play, stop, and pause. CLVS, however, does not allow any communication between the client and the video source. Therefore, the viewer cannot use the built in functions of RTSP. This is not problematic though. The main purpose of RTSP is to relieve the server of extra obligations when a video is paused or stopped. Since CLVS happens in the cloud, extra pulls by more clients are automatically scaled. Also, a benefit to CLVS is that it can continue to upload video segments at low cost. CLVS also avoids the overhead of some RTSP operations, such as: setup, pause, teardown, describe, announce, set parameter, etc. Therefore, the inability of CLVS to provide RTSP operations has a negligible effect.

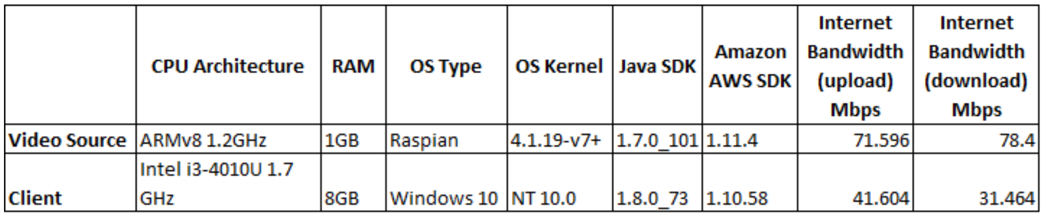
CLVS makes no attempt to debug or optimize client performance through live monitoring. This creates limitations on performance enhancements from performance monitoring protocols like RTCP. Therefore, issues such as packet loss, are not considered. Though necessary for modern video streaming QoS, transcoding is also not considered with CLVS.

3.7 Limitations

There are some drawbacks to CLVS. First, there are no elements of adaptive streaming. A client who uses a large TV screen will receive the same data as a client who views the video on a smart phone, which results in worse throughput and higher costs. Second, the compression algorithm is not optimal. CLVS uses intra-frame compression, whereas all modern video streaming use inter-frame compression. Therefore, the quality of the video is going to be worse or the size of the video is going to be much larger when compared to other video streaming techniques. Along with poor compression, CLVS makes no attempt to include sound. Third, there is no flexibility when choosing protocols. CLVS protocols are limited to the cloud storage protocols used by the provider. For instance, live video streaming typically uses UDP along with RTSP [32], but S3 is limited to the protocols mention in Section 3.4. Finally, there are no ways to mitigate packet loss. As mentioned in Section 2.1, streaming protocols are able to hide packet loss through FEC and interleaving. In CLVS, however, a network issue that occurs during a segment download can result in an entire segment loss rather than a minor frame loss. Some of these issues can be solved, while others are inevitable. In this work, we propose a new approach of live video transfer using cloud storage. We verify the feasibility, design efficiency, and analyze its cost and performance.

4 Evaluation

For a performance comparison, we used Wowza Streaming Engine as recommended by Amazon [34]. This solution does not require the setup and purchase of server. Rather, an EC2 sever is rented through an Amazon Machine Image (AMI). An AMI consists of a template for a root volume including an operating system, an application server, and applications. It also contains launch permissions that control which AWS accounts can use the AMI to launch instances. In this case, a pre-built Wowza Streaming Engine AMI is used and acts as a server between a video source and its viewers.



	CPU Architecture	RAM	OS Type	OS Kernel	Java SDK	Amazon AWS SDK	Internet Bandwidth (upload) Mbps	Internet Bandwidth (download) Mbps
Video Source	ARMv8 1.2GHz	1GB	Raspian	4.1.19-v7+	1.7.0_101	1.11.4	71.596	78.4
Client	Intel i3-4010U 1.7 GHz	8GB	Windows 10	NT 10.0	1.8.0_73	1.10.58	41.604	31.464

Figure 4.1: CLVS environment details

The EC2 machine chosen for these experiments is a m3.xlarge instance type and contains 4 virtual CPUs, 15 GiBs of memory, two SSDs with 40 GBs storage, and ”high” networking performance (max bandwidth of 500 Mbps and expected throughput of 62.5 MB/s). FFMPEG is used as the encoder for the video source, which uses H.264 compression. RTMP was selected as the main transfer protocol since it generally has lower live stream delays than other HTTP-based protocols [35]. The Wowza Streaming Engine AMI instance used in the evaluation will further be referred to as Wowza. Specifications used for CLVS can be viewed in Figure 4.1.

4.1 Measurements

Testing video quality is different than measuring network quality [36], and there are no universal standards in place for doing so. The goal of this project, again, is to assess the practicality and efficiency of CLVS compared to other modern live event streaming methods (in this case Wowza).

The following contains a list of metrics chosen to measure video quality [37] [38] [39]:

- *Bit Rate*: Indicates how many bits of video can be transmitted over a specific period of time.
- *Connect Time*: Measures the amount of time elapsed between the initial request for data by the media player and the start of buffering. This includes DNS lookup, metafile actions and resolution, server/player handshakes, and the transport of the first byte of data to the player.
- *Buffer Time*: Measures the amount of time used to build up the initial buffer. The time between the first byte downloaded to the playback of the video.
- *Rebuffer Event*: When video playback is delayed and more data is needed to view the video.
- *Rebuffer Time*: The total amount of time spent during all rebuffer events.
- *Lag Length*: This includes the buffer time, plus connect time, plus rebuffer time.
- *Play Length*: The amount of time the video is played without lag.
- *Lag Ratio*: The percentage of buffering versus total viewing time. Represents the time spent buffering while viewing the video.
- *Delay*: The length of time between the recorded event and when it is viewed.
- *Cost*: The total monetary cost of running the video stream.

There are three main measurements of key importance for testing a video stream - video quality, delay, and cost. Video quality consists of the smoothness of the playback and the different visual characteristics such as color depth, sharpness, etc. The delay is the time difference between the upload and download of a segment (how far behind the viewer is from the live event). And cost is the actual monetary cost of using the video streaming service. These measurements are used to compare the performance between CLVS and Wowza.

Video quality consists of both lag ratio and visual quality. Visual quality can be difficult to measure [40] and there are many considerations. High visual quality generally indicates strong compression algorithms like h.264. Here, however, a simple JPEG compression is being used. Since compression efficiency is not the goal of this project, the more appropriate way to measure video quality is through bit rate. Bit rate measures the total amount of data transmitted rather than the visual quality of the playback. This is based on the assumption that if CLVS uses a similar compression algorithm as its competitor, it will achieve the same visual quality without higher data usage.

In order to capture delay, both the video source and client had to be synchronized. The NTP Pool Project [41] maintains virtual servers that allow other computers to sync to one of their server's time clock. Both video source and client devices were synced to the same ntp virtual server before uploading or downloading any video segments.

To measure the cost of using CLVS, S3 provides a pricing index as shown in Figure 3.5. The cost of CLVS depends solely on S3 and its region, which in this case is US-West-2 (Oregon). The cost of using Wowza is dependent on the Wowza Streaming Engine license agreement and chosen EC2 instance [43].

4.2 Testing Process

In order to measure the metrics mentioned above (Section 4.1), separate approaches were needed for CLVS and Wowza. Since all of the CLVS code is available in Java, it was simple to implement each measurement within the code itself. Wowza, in contrast, required more unconventional means to acquire its metrics. Wowza is proprietary software so its code is not accessible and its extensible modules are limited. In addition, monitoring provided by Wowza only shares basic information: CPU utilization, memory use, network bandwidth, number of connections, and available disk space. Such information as lag ratio and delay was not available.

Video streaming quality metrics can be measured effectively through direct code implementation in CLVS. Two bit rate measurements are taken - one from the video source and the other from the client. A collection of bit rates are stored in a buffer and an average is taken among these values. Each bit rate is recorded after a certain number of segment plays, which is based on a parameter. Connect time starts as one of the first lines of code in the program and stops right before the first image is displayed. Buffer time is collected between the time when a segment ends and another segment begins. Play length starts right before the first frame of a segment is shown and ends when after its last frame is displayed. Delay is $currenttime - starttime - segmenttimestamp$. As previously mentioned in Section 4.1, both devices are synchronized to the same NTP time clock before this value is calculated.

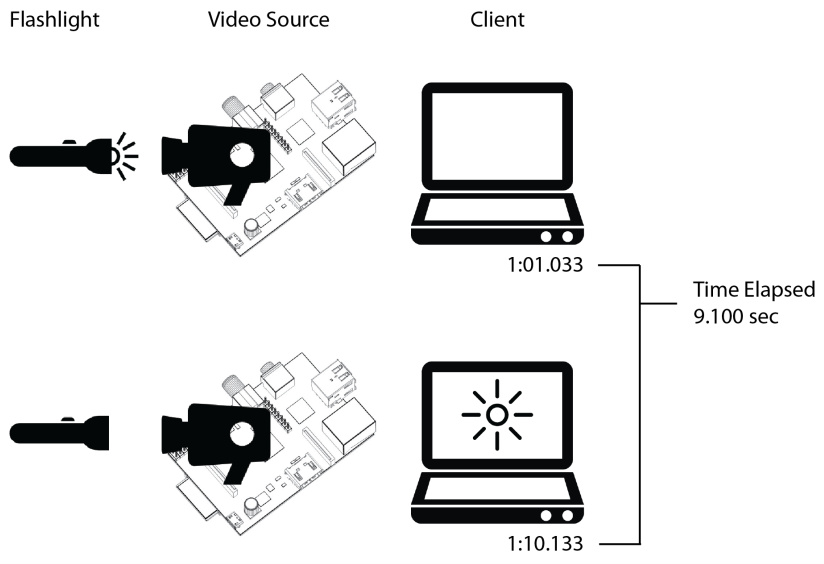


Figure 4.2: Wowza delay measurement methodology

Two different methods are used to gather metrics from Wowza. The first method uses Wireshark to measure client bit rate and duration. Duration is the total amount of time when packets were sent and received between Wowza and the client. Server bit rate was taken directly from FFmpeg. The second method uses a video camera that records both the video source and client device during each test. This method acquires the play length and delay measurements. Play length starts when the first frame is present and ends at the last frame. Buffer length then becomes $duration - playlength$. To measure delay, a light is flashed every minute on the source device. The time between the flash and its appearance on the client device becomes a record of delay (see Figure 4.2). This method produces accurate results within $1/30th$ of a second. An average is taken for each test based on ten minutes of recording.

4.3 Stability

There were two tests used to measure the behavior and stability of CLVS. The first is a longevity test and the second is a segment length variation test. Both tests use the same visual quality parameters, devices, and network connection.

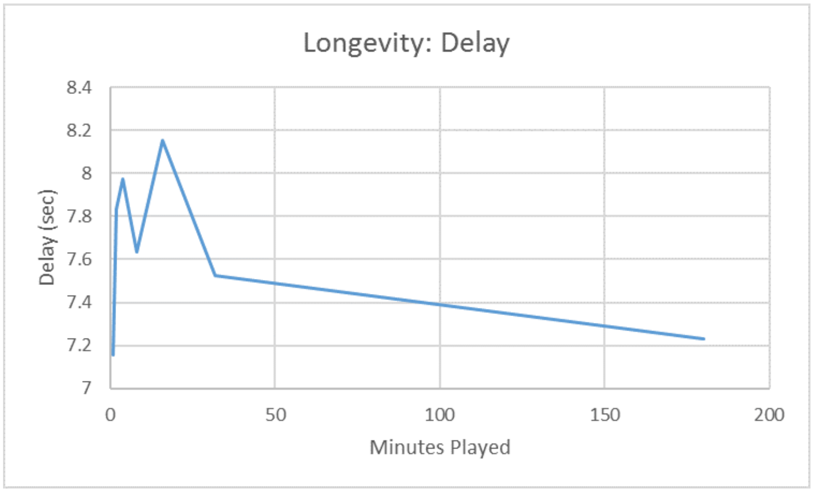


Figure 4.3: CLVS longevity test delay results with a 4 sec segment length

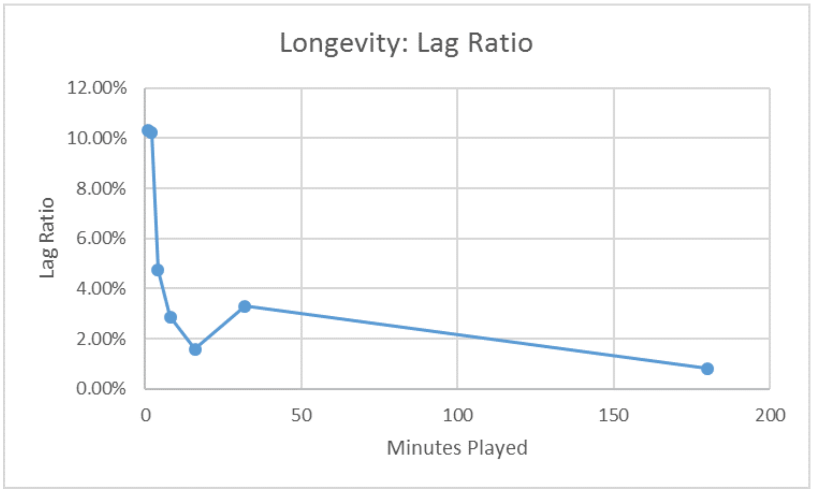


Figure 4.4: CLVS longevity test lag ratio results

The longevity test is used to see how CLVS performs when it runs for different lengths of time and uses a 4 second segment length. An average was taken of 20 tests for 1-2 minutes, 5 tests for 4-8 minutes, 3 tests for 16 minutes, and 1 test for 32-180 minutes. The trend indicates that the time spent buffering is reduced the longer the video stream runs (see Figure 4.4). This behavior could be attributed to initial startup costs. When first connecting to S3, there are more timeouts during initial downloads, which causes more buffer events and segment drops. Therefore, the shorter the video stream the more extravagant the lag ratio. There is also a spike for the stream length of 32 minutes. This is because only one test was used for this value, and its results are attributed to unexpected network issues. The delay results in Figure 4.3 are fairly consistent, with the largest variation being one second. Minor variations in delay can be attributed to network behavior.

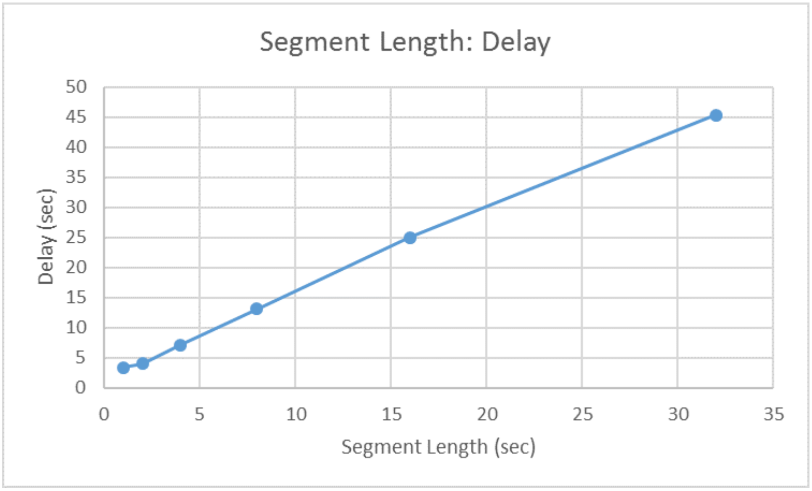


Figure 4.5: CLVS segment test delay results

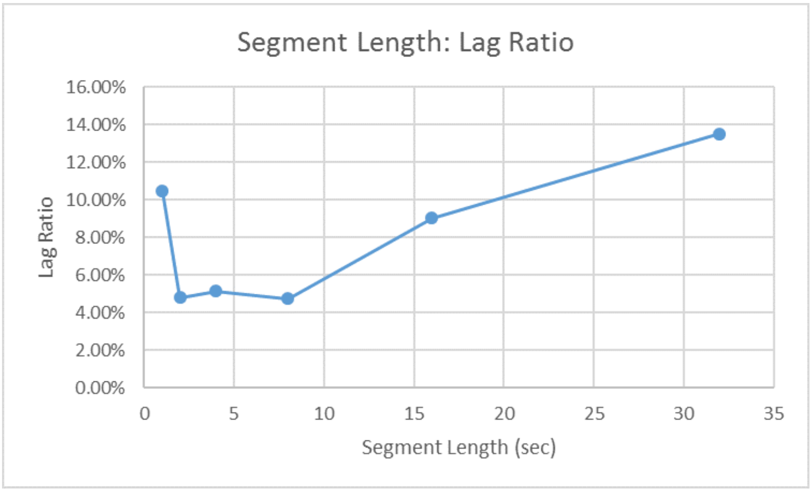


Figure 4.6: CLVS segment test lag ratio results

The segment length variation test is used to determine the behavior of CLVS when using different segment lengths. An average was taken of 10 tests for each segment length value and each test was run for 4 minutes. The results show that the ideal segment length is between 2-8 seconds, where the lag ratio is at its lowest (see Figure 4.6). When the segment length is too short, there is too much overhead to be effective. The video source is constantly recording and sending data. When it takes longer to send a segment than to record a segment, there is inherent rebuffer events before the client even downloads the segment. This results in many constant and consistent segment drops when viewing the video. If the segment is too large, it takes longer to load each segment. There are no segment drops, but there is a large initial delay. Also, any network issues that require additional attempts to download a segment would be more detrimental with larger segments. There is a very linear relationship between segment length and delay (see Figure 4.5). This is expected, since the $delay = segmentlength + uploadtime + downloadtime$.

The longevity test shows system stability, while the segment variation test illustrates the effect of the segment length parameter to lag ratio. The segment test also asserts the assumed delay behavior. These tests indicate that CLVS can run for long periods of time and that segment lengths of 2-8 seconds provide the least amount of lag ratio. These tests were used to indicate the ideal parameters for CLVS in order to compare it to another modern video streaming methodology.

4.4 Performance Comparison

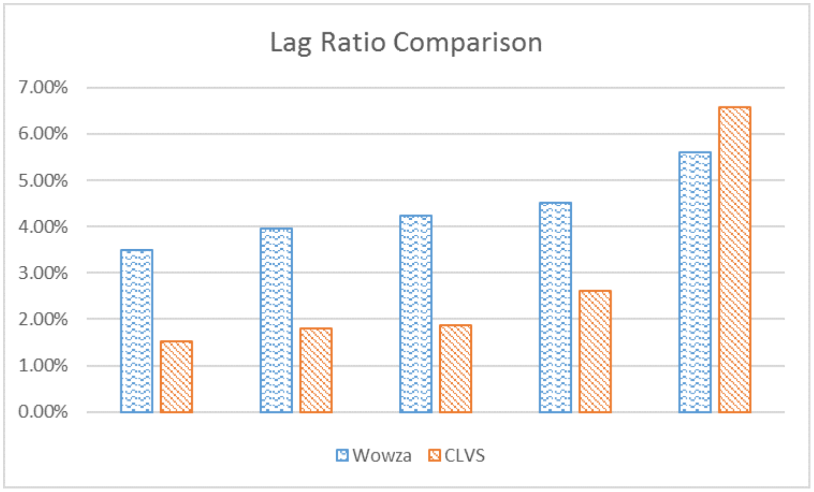


Figure 4.7: CLVS and Wowza comparison results (Lag Ratio)

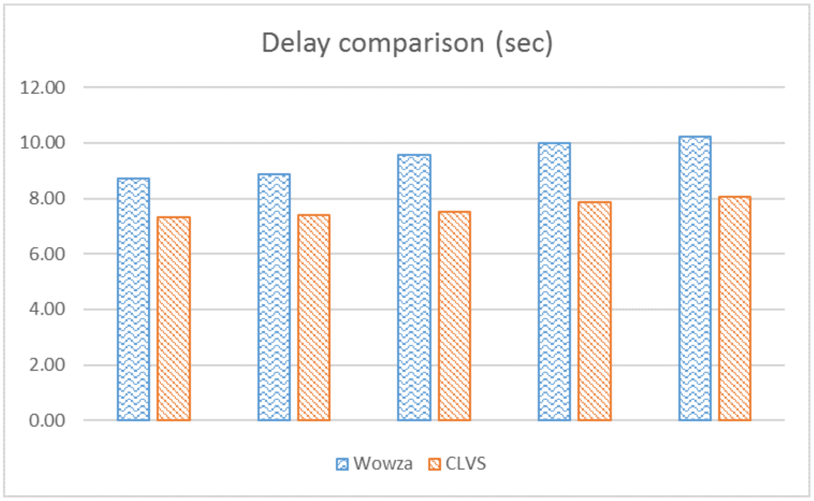


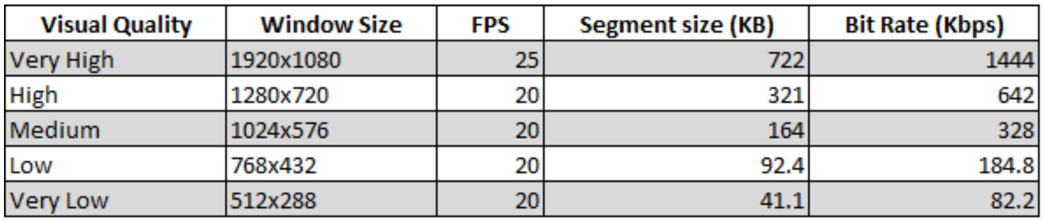
Figure 4.8: CLVS and Wowza comparison results (Delay)

The results in Figure 4.7 and Figure 4.8 show CLVS outperforming Wowza in most cases with respect to delay and lag ratio. The comparison takes the best five results from ten Wowza tests and compares them to all five results taken from a CLVS longevity tests (segment length = 4sec, test length = 8 min). On average, CLVS has a delay 3 seconds less than Wowza. This could be attributed to the initial overhead of using the RTMP. As mentioned before in Section 2.1, RTMP has to open many channels to effectively transmit media. Also, RTMP converts media into swf files so that it can be viewed in Adobe Flash. In addition to better delay performance, CLVS also has a 1.5% lower lag ratio than Wowza on average. In regards to overall performance, CLVS is faster and has less jitter than Wowza.

CLVS performs better despite many favorable testing characteristics setup for Wowza. First, Wowza used a lower bit rate than CLVS. This means that less information had to go from the video source to the client. The average bitrate for Wowza was 188 Kbps, while the average client bitrate for CLVS was 319 Kbps. Also, CLVS was able to receive a higher percentage of data than Wowza. The CLVS client bitrate was 96.84% of its video source, while the Wowza client bitrate was 90.60%. This means that CLVS was able to receive and use more data sent from the video source than Wowza. Second, the lag ratio for Wowza should be higher since it does not account for rebuffer events. Rebuffer length is determined by $duration - playlength$, but does not include rebuffer events that occur during video playback. Third, the comparison results use Wowza's top five best results whereas CLVS only had five results in total. Though not deterministic, this comparison does favor Wowza. Also, CLVS was only tested for eight minutes instead of ten. In the longevity tests, lag ratio generally improves with longer run times. Again, not deterministic but still favorable. In summary, the overall Wowza tests sent and received less data, did not account for buffer events, and has favorable bias when compared to CLVS. Despite these favoritisms, CLVS still performed better overall.

4.5 Cost

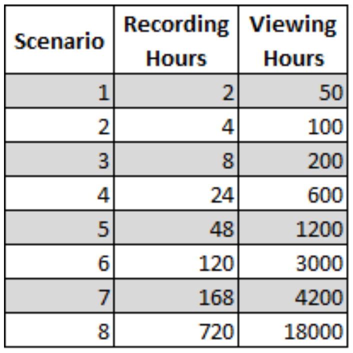
The cost of using Wowza is \$0.76 per instance per hour plus \$15 per month [43]. The cost of using CLVS, however, is more dynamic. For further reference, all prices associated with CLVS are based on using the US-West-2 (Oregon) pricing index (see Figure 3.5). These cost comparisons are based on the same visual qualities, with CLVS using a segment length of four to match the parameters use in Section 4.4.



Visual Quality	Window Size	FPS	Segment size (KB)	Bit Rate (Kbps)
Very High	1920x1080	25	722	1444
High	1280x720	20	321	642
Medium	1024x576	20	164	328
Low	768x432	20	92.4	184.8
Very Low	512x288	20	41.1	82.2

Figure 4.9: Video quality specifications

The first cost comparison checks different video qualities (Figure 4.9) under different scenarios (Figure 4.10). Video qualities are important to distinguish since S3 pricing is partly based on the amount of data that goes out to the Internet (Data Transfer OUT). Wowza, on the other hand, does not base its price on any form of bandwidth usage. Rather, Wowza provides a certain amount of bandwidth (roughly 500 Mbps) and will not scale beyond that limit. S3, however, can scale on demand up to an unspecified amount. Most claims pertaining to S3 scalability base its effectiveness on the amount of GET requests. Amazon claims that S3 can handle up to 800 requests effectively [44], while some S3 customers are said to be performing thousands of requests per second [45]. All selected scenarios



Scenario	Recording Hours	Viewing Hours
1	2	50
2	4	100
3	8	200
4	24	600
5	48	1200
6	120	3000
7	168	4200
8	720	18000

Figure 4.10: Test scenarios based on longer recordings with proportional views

use proportional recording and viewing hours (1:25).

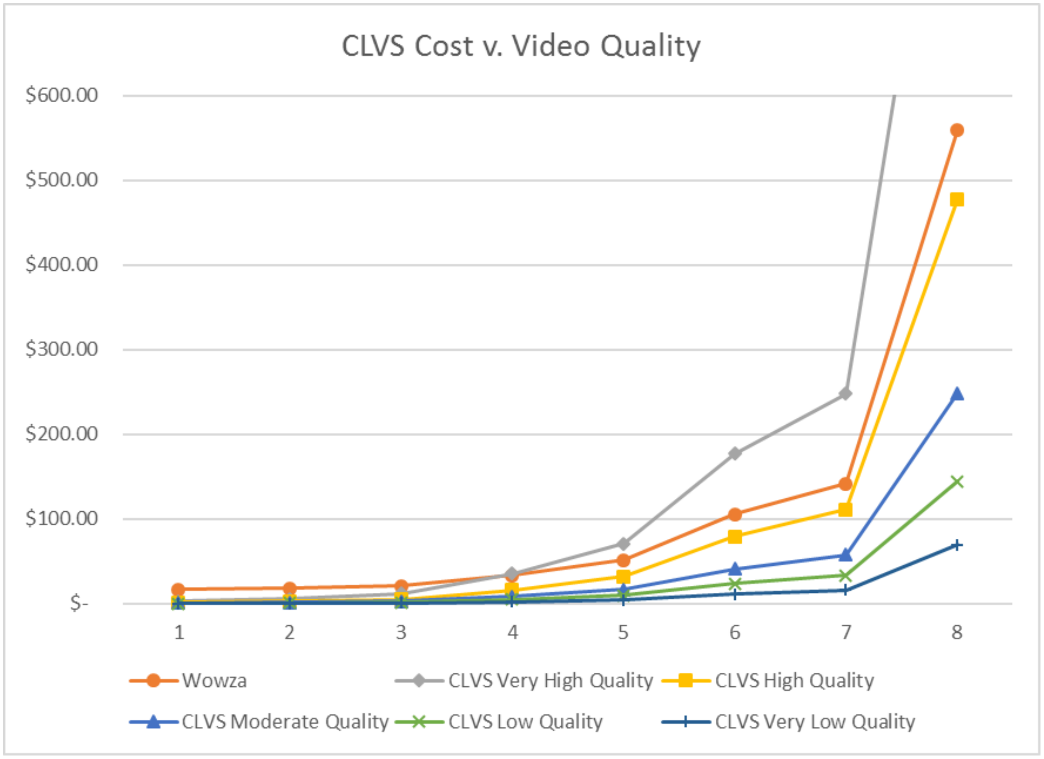


Figure 4.11: CLVS and Wowza cost comparison using different video qualities

The cost comparison results shown in Figure 4.11 provides insight into the two different cost models. The Wowza cost remains the same no matter the video quality. CLVS, however, is vastly different depending on the bit rate. For all video qualities, CLVS is less expensive than Wowza for scenarios 1-3. This is because Wowza requires the licensing fee of $15 dollars a month, which itself outweighs the cost of using S3 under these conditions. For scenarios 4-8, the very high quality video stream running on CLVS causes its price to rise above Wowza at nearly double the cost by scenario 8. The rest of the video streams, however, are less expensive to run using CLVS under all chosen scenarios.

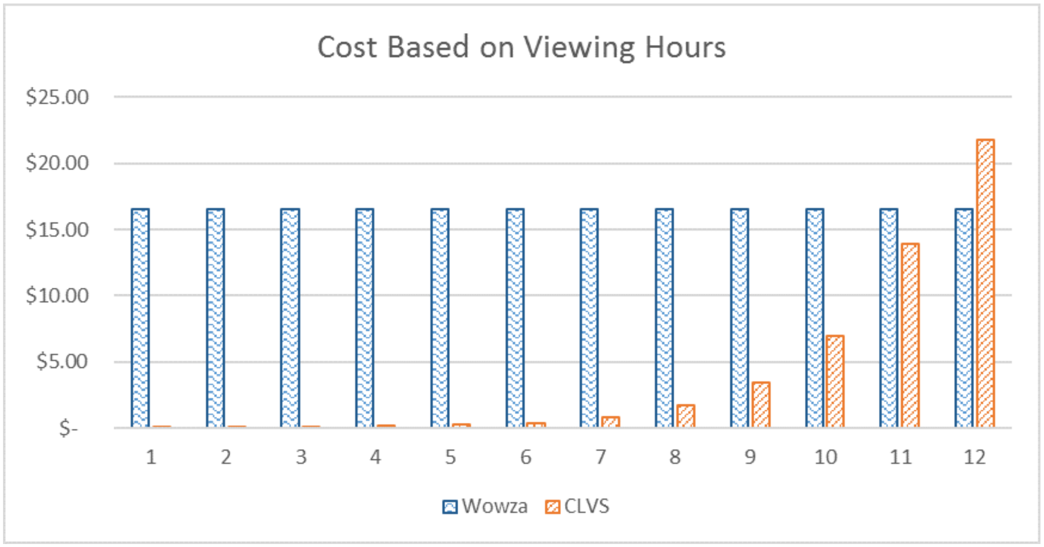
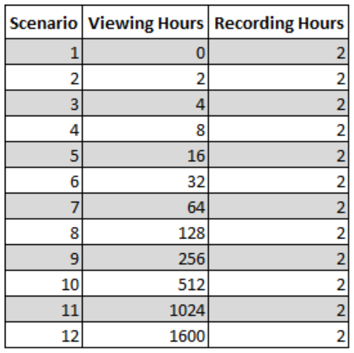


Figure 4.12: CLVS and Wowza cost comparison using scenarios from Figure 4.13

The next cost comparison looks at the total number of viewing hours, while using the same number of recording hours in the previous example (see Figure 4.13). This comparison is based on medium video quality. Again, the price for Wowza remains constant and its higher costs are primarily attributed to its licensing agreement as seen in Figure 4.12. CLVS costs remain much lower than Wowza until scenario 12. This comparison illustrates that the cost of CLVS primarily depends on the amount of viewing hours, whereas the price of Wowza is based on recording hours.



Scenario	Viewing Hours	Recording Hours
1	0	2
2	2	2
3	4	2
4	8	2
5	16	2
6	32	2
7	64	2
8	128	2
9	256	2
10	512	2
11	1024	2
12	1600	2

Figure 4.13: Test scenarios based on higher views only

$$(SegmentsPerHour) * ((RecordingHours * PUTRequestCost) + (ViewingHours * GETRequestCost) + (DataTransferOUTCost * SegmentSize)) + (StorageCost * MaxSegmentsSaved * SegmentSize)$$

Figure 4.14: CLVS cost equation

It is difficult to say whether CLVS or Wowza will be more cost-effective without knowing how it will be used. CLVS is better for situations where there will be constant recording and less viewing. Wowza can provide better video quality and allow a high number of users to view the stream for cheaper. CLVS should be able to scale more viewers at higher bit rates, but the cost can be much more expensive. Therefore, one method is not always better than the other. To help determine if CLVS is applicable for a given scenario, the cost can be accurately predicted using the formula in Figure 4.14. CLVS provides a different cost direction and can be more effective depending on the scenario.

5 Discussion

Though CLVS offers good overall performance, it does not replace the existing model for live video streaming. Rather, CLVS offers a great alternative in different situations. For instance, CLVS works very well for security monitoring, small events, business interests (child care, ski resorts, zoos, etc), and personal interests. CLVS does not necessarily work well for live communication, large events, and widely viewed events. This is the same concept as polyglot persistence for data storage.

Live communication does not work well with CLVS. Generally, live video streaming for communication purposes use UDP as its main transport protocol. Skype for Business uses the H.323 protocol, which is a combination of TCP and UDP [7]. In this instance, UDP is tasked with the responsibility of transporting audio and video data, while TCP handles time insensitive operations. TCP is not ideal for media transfer, in this case, since its main priority is guaranteed service. UDP, on the other hand, cares most about fast delivery. Since CLVS uses cloud storage, it has to use the TCP protocol. In addition to its rigid protocol selection, the cost of using CLVS increases exponentially as the stream gets closer to real time. This is due to the increasing number of PUT and GET requests. For example, two second segments cost twice as much as four second segments, two second segments cost twice as much as one second segments, one second segments cost twice as much as half second segments, and so on. Since segment length can be infinitely close to zero, the price can theoretically be infinite! This, however, is impossible in the real world. The segment length is ultimately bound by hardware and networking capabilities. The camera can only record up to a certain FPS and cloud storage is limited by its communication and IO performance. UDP streaming protocols like RTP are best suited for live communication.

Events that require a large audience may or may not work well with CLVS. As previously mentioned, S3 can handle up to 800 GET requests before experiencing performance concerns [44]. This can be resolved by either sending a support case to AWS or by using a CDN. CDNs are typically used to stream live events to a large audience. Akamai, a CDN provider, distributed the content for the 2016 Olympics and the 2016 Presidential Elections [46]. Amazon CloudFront is a CDN provided by AWS. In order to retrieve the video stream, CloudFront has to grab the video from either S3 or an existing HTTP server. In other words, there is potential for CLVS to use CloudFront with S3. Another concern, however, is transcoding. When streaming video to a large audience, transcoding becomes an asset for relieving congestion. The CLVS model, however, does not currently support transcoding.

6 Future Work

CLVS shows great promise as a live streaming competitor, but there are still many improvements to be made. Though CLVS is a prototype to show its potential streaming advantages, certain characteristics should be included with the current design. For instance, CLVS does not currently support sound. To avoid file duplication, sound clip data should be included in the data buffer within the video segment. The implementation of sound, though, will depend on future compression integration. Intra-frame compression is not ideal for modern video streaming. Therefore, an optimal compression scheme (perhaps h.264) should be implemented as well. Any attempt to do so should avoid writing to disk on either the video source or client device.

S3 requires a user to provide credentials in order to access data. At present, every user of CLVS has to provide such credentials in the form of a file. Another safeguard is the TLS 1.2 protocol that encrypts the data to avoid unwanted access from anyone monitoring the stream. Both of these processes provide good application security. In order for an unwanted user to access the stream, he or she would have to access the actual credentials file located on another users machine. To make a video stream available to the public, however, would require some extra work since the client has to ensure that their credentials are not public. One way is to stream to a URL. A primary benefit to using CLVS is that no server is required, so using a server to host a URL is counter productive. S3, though, can host a website. Therefore, one possible solution that needs further exploration is hosting CLVS through an S3 URL.

Streaming video through mobile devices is very common. For both the customer and the video stream provider, a transcoder is used to reduce bandwidth costs. This way mobile users receive the video faster with little noticeable difference in video quality and stream providers can reduce the congestion on their servers. In order to use a transcoder with CLVS, though, multiple recordings have to occur on the video source. This could possibly increase delay since the video source has to perform additional work. This process will also cause multiple uploads for each single video segment upload, Therefore, the number of PUT requests will increase by a factor of the desired number of video formats. For instance, there could be a video format designed for mobile, tablet, PC, and TV. Each format requires an upload to S3, so the original PUT requests cost will increase by a factor of 4. Transcoding could potentially be a parameter in CLVS where the client has the option whether or not to transcode their video stream. Despite higher PUT requests costs, the overall cost of using CLVS could potentially be cheaper. This is because the data transfer OUT costs would be lower for any user who downloads at lower bitrates.

FEC is not possible since CLVS cannot control how packets are sent and received with S3. However, it is possible to accommodate timeout recovery to reduce jitter on a video stream. If a timeout occurs during a video segment download, CLVS skips that segment and waits for the next one to upload. This helps reduce delay, but the segment drop is very noticeable. One way to approach this issue is to apply time correction. For instance, three seconds has elapsed since the last video segment stopped playing. With a segment length of eight seconds, 3/8 of the next video segment should have played. After the next video segment is downloaded, its index position could skip 3/8 ahead of the total index. For instance, where segment length is 8 and FPS is 20, the total number of frames is 160. $160 * (3/8) = 60$, therefore the next video segment should start at frame 60 instead of being dropped entirely.

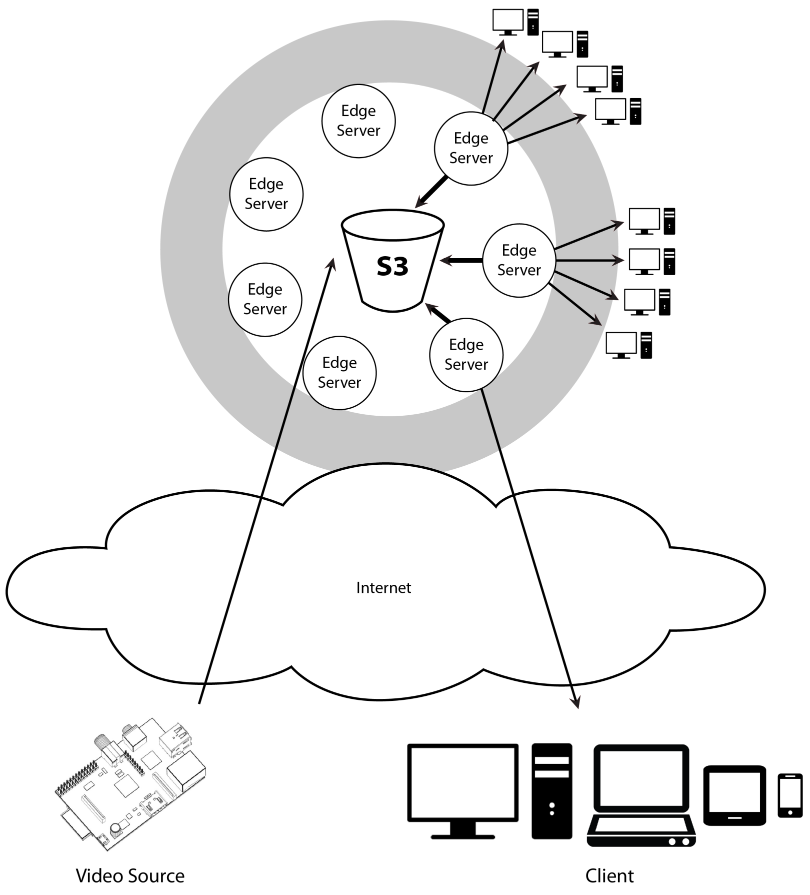


Figure 6.1: Potential CLVS CloudFront Model

Amazon CloudFront can directly access S3. Therefore, CLVS has the potential to run on a CDN. Figure 6.1 shows how communication could work in CLVS with CloudFront integration. A major benefit to using CLVS with CloudFront is cost. The pricing model for CloudFront is based on the amount of data transferred out from their edge servers. The data transfer OUT cost for S3 becomes zero when using CloudFront, and is instead based on the CloudFront pricing model. Therefore, regardless of the video streaming technique (provided transcoding and compression are comparable) the cost will be lower when using CLVS. This is because other video streaming services use the same cost plus the cost of using a CDN. CLVS, on the other hand, can eliminate its data transfer OUT cost. In Section 4.5, data transfer OUT is the most significant expense for handling large viewing hours. This cost is neglected by S3 when using CloudFront, meaning that this high cost property will be lower to any other service using a CDN. The rest of its costs will remain lower than modern streaming services.

There are many other potential opportunities to expand CLVS. First, it would be simple to save backups of images or video since the stream already resides within cloud storage. Second, cloud cameras could use CLVS and avoid tracking customer data. As mentioned in Section 2.5, cloud cameras require users to have personal accounts. This provides an opportunity for users to add new features, receive image alerts, and manage their subscription. Rather, cloud cameras could run CLVS based on a S3 credentials and bucket info. Other services could be defined in parameters available to users. There are a number of ways to expand CLVS to adapt to the modern marketplace.

The opportunities for CLVS in the market are vast. Currently, the CLVS model can be used by small businesses. Daycare centers could provide video streams to parents whom want to know how their child is doing. Business owners could use CLVS to keep track of business operations while he or she is away. Individuals could setup CLVS at home for security or to keep an eye on their pets. At this point, any situation that has many recording hours than viewing hours can benefit by using CLVS. With a CloudFront integration, this opportunity expands to large scale serverless video streaming. CLVS accommodates the modern marketplace, as well as opens up more avenues for innovation.

7 Conclusion

Live video streaming is growing in popularity as the streaming video industry has become a major cable competitor. CLVS provides a very efficient method for streaming live video and is proven to be effective, stable, and inexpensive under different scenarios. Along with cost savings, CLVS removes the need for third party software, as well as license agreements, to provide users with a means to stream their video without subscriptions to stream providers. CLVS can be effective for personal use, small company promotions, security, and potentially large distributed streaming when paired with a CDN. As more viewers migrate to live video streaming, CLVS can play a pivotal role in providing a platform for more inclusion and opportunity.

Vita

Author: Ryan J. Babcock

Place of Birth: Seattle, Washington

Undergraduate Schools Attended: Western Washington University

Eastern Washington University

Degrees Awarded: Bachelor of Arts in Fine Art, 2008, Western Washington University

References

[1] N. McAlone, “Services like netflix and hulu are growing much faster than cable,” apr 2016. [Online]. Available: http://www.businessinsider.com/growth-of-streaming-services-outpacing-traditional-cable-2016-4

[2] A. Gomez, “Live streaming vs video on demand: Why brands need to consider both,” mar 2016. [Online]. Available: http://tubularinsights.com/live-streaming-vs-video-on-demand/

[3] I. Ustream, “Find the best video streaming platform: Compare pricing & options,” 2017. [Online]. Available: https://www.ustream.tv/platform/plans?itm_source=home_header&itm_medium=onsite&itm_content=pricing&itm_campaign=top_header

[4] DaCast, “Dacast homepage,” 2017. [Online]. Available: http://www.dacast.com/

[5] Livestream, “Livestream platform plans,” 2017. [Online]. Available: https://livestream.com/platform/pricing

[6] A. Inc., “Http live streaming.” [Online]. Available: https://developer.apple.com/streaming/

[7] “Part 3: Networks and protocols used by skype for business 2015,” 2017. [Online]. Available: http://www.c21video.com/microsoft_lync/skype_for_business_network_protocols.html

[8] G. McGath, “Basics of streaming protocols,” 2013. [Online]. Available: http://www.garymcgath.com/streamingprotocols.html

[9] M. Yan, “How does video streaming work? [part 3: Http-based adaptive streaming],” June 2012. [Online]. Available: http://mingfeiy.com/adaptive-streaming-video-streaming

[10] “Real-time messaging protocol,” December 2016. [Online]. Available: https://en.wikipedia.org/wiki/Real-Time_Messaging_Protocol

[11] N. Drakos, “Rtp,” 1994. [Online]. Available: http://www4.cs.fau.de/Projects/JRTP/pmt/node30.html#SECTION0

[12] H. Schulzrinne, “Some frequently asked questions about rtp,” October 2013. [Online]. Available: http://www.cs.columbia.edu/~hgs/rtp/faq.html

[13] "Http adaptive streaming," March 2013. [Online]. Available: https://github.com/rabit/wiki/blob/master/HTTP_Adaptive_Streaming.md

[14] J. Ozer, "What is adaptive streaming?" April 2011. [Online]. Available: http://www.streamingmedia.com/Articles/Editorial/What-Is-.../What-is-Adaptive-Streaming-75195.aspx

[15] "What is transcoding and why is it critical for streaming?" March 2015. [Online]. Available: https://www.wowza.com/blog/what-is-transcoding-and-why-its-critical-for-streaming

[16] AWS, "Architecting for the cloud: Aws cloud best practices," February 2016. [Online]. Available: https://d0.awsstatic.com/whitepapers/AWS_Cloud_Best_Practices.pdf

[17] W. Vogels, "All things distributed," October 2007. [Online]. Available: http://www.allthingsdistributed.com/2007/10/amazons_dynamo.html

[18] "Amazon simple storage service (s3) faqs." [Online]. Available: https://aws.amazon.com/s3/faqs/

[19] "Video compression standards," 2014. [Online]. Available: https://www.mistralsolutions.com/video-compression-standards-pros-cons/

[20] S. P. S. Anurag Jagetiya, "A seminar on mpeg1 and mpeg2," January 2015. [Online]. Available: https://www.slideshare.net/anuragjagetiya/mpeg-video-compression-standard

[21] "An explanation of video compression techniques."

[22] D. Karakasilis, F. Georgatos, L. Lambrinos, and T. Alexopoulos, "Application of live video streaming over grid and cloud infrastructures," in *2011 IEEE 11th International Conference on Computer and Information Technology*, Aug 2011, pp. 379–383.

[23] X. Li, M. A. Salehi, and M. Bayoumi, "Vlsc: Video live streaming using cloud services," in *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom)*, Oct 2016, pp. 595–600.

[24] C. McDonald, “iphone windowed http live streaming using amazon s3 and cloudfront proof of concept,” July 2009. [Online]. Available: http://www.ioncannon.net/programming/475/iphone-windowed-http-live-streaming-using-amazon-s3-and-cloudfront-proof-of-concept/

[25] “Dropcam case study,” 2017. [Online]. Available: https://aws.amazon.com/solutions/case-studies/dropcam/

[26] “Setting up your ring video doorbell in the ring app,” April 2017. [Online]. Available: https://support.ring.com/hc/en-us/articles/115001773266-Setting-Up-Your-Ring-Video-Doorbell-In-the-Ring-App

[27] “Limited and full accounts for your customers,” 2017. [Online]. Available: https://intercom.help/angelcam/for-partners/billing/limited-full-accounts-for-your-customers

[28] “Aws case study: Y-cam solutions,” 2017. [Online]. Available: https://aws.amazon.com/solutions/case-studies/y-cam/

[29] “Case study,” July 2014. [Online]. Available: https://www.security.honeywell.com/documents/L_DVCAPCS_D.pdf

[30] “Netflix case study,” 2016. [Online]. Available: https://aws.amazon.com/solutions/case-studies/netflix/

[31] S. Gibbs, “Google’s new nest cam is always watching, if you let it into your home,” June 2015. [Online]. Available: https://www.theguardian.com/technology/2015/jun/18/googles-nest-cam-always-watching-live-streaming-video

[32] S. Fu, “Why does netflix use tcp and not udp for its streaming video?” 2016. [Online]. Available: https://www.quora.com/Why-does-Netflix-use-TCP-and-not-UDP-for-its-streaming-video

[33] “Amazon s3 pricing,” 2017. [Online]. Available: https://aws.amazon.com/s3/pricing/

[34] “Amazon cloudfront media streaming tutorials,” 2017. [Online]. Available: https://aws.amazon.com/cloudfront/streaming/

[35] F. Altomare, “Streaming: Adobe rtmp explained,” March 2015. [Online]. Available: http://www.globaldots.com/streaming-adobe-rtmp-explained/

[36] S. I. ULC, “Video quality of experience: Requirements and considerations for meaningful insight: An industry whitepaper,” 2016, http://www.vlinklive.com/explore-live-streaming/frequently-asked-uestions/what-is-live-video-streaming-or-livecasting/.

[37] S. Berger, “Measuring and monitoring streaming media quality,” aug 2004. [Online]. Available: http://www.streamingmedia.com/Articles/Editorial/Featured-Articles/Measuring-and-Monitoring-Streaming-Media-Quality-64648.aspx

[38] T. Hinds, “Top 5 metrics for streaming video performance,” Mar 2014. [Online]. Available: http://www.neotys.com/blog/top-5-metrics-for-streaming-video-performance/

[39] B. Phillips, “Live streaming stats: Which metrics matter?” Nov 2016. [Online]. Available: https://blog.peer5.com/live-streaming-stats-which-metrics-matter/

[40] N. Dye, “How-to: Best practices for measuring performance of streaming video,” 2016. [Online]. Available: https://link.brightcove.com/services/player/bcpid897709167001?bckey=AQ~~,AAAAADEURYw~,kpjcfLGBbVfV_nqUUCkA1vYpmRTGak7R&bctid=2016262274001

[41] B. Hansen, “pool.ntp.org: public ntp time server for everyone,” 2017. [Online]. Available: http://www.pool.ntp.org/en/

[42] “Simple monthly calculator,” 2017. [Online]. Available: https://calculator.s3.amazonaws.com/index.html

[43] “Wowza streaming engine 4: Pro edition (hvm),” 2017. [Online]. Available: https://aws.amazon.com/marketplace/pp/B012BW3WB8?qid=1476902318946&sr=0-1&ref_=srh_res_product_title/ref=_pntr_web_deo_opt

[44] “Request rate and performance considerations,” 2017. [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/dev/request-rate-perf-considerations.html

[45] J. Barr, “Amazon s3 performance tips & tricks - seattle s3 hiring event,” March 2012. [Online]. Available: https://aws.amazon.com/blogs/aws/amazon-s3-performance-tips-tricks-seattle-hiring-event/

[46] “You get one shot with live streaming,” 2017. [Online]. Available: https://www.akamai.com/us/en/media-and-delivery/live-events.jsp