Abstract— In recent years, the consumption of video content over the Internet has increased exponentially, with IP video traffic making up more than two thirds of all consumer traffic. Prior to establishing a standard, video delivery over the best effort Internet was carried out by proprietary protocols like Apple’s HTTP Live Streaming (HLS), Adobe's HTTP Dynamic Streaming and Microsoft's Smooth Streaming. Developed under the Moving Picture Experts Group (MPEG), Dynamic Adaptive Streaming over HTTP (DASH) is fast becoming the globally preferred standard for video streaming because it allows reuse of the widely deployed standard HTTP servers/caches. The repository used to store the video data consists of multiple copies of the same video encoded at different quality rates which can be used to adjust the user’s stream depending on network conditions. However, multiple data streams lead to inefficient utilization of data center resources, as the lowest resolution video data, which is common to each stream, occupies a major chunk of the storage space. This redundancy can be eliminated by implementing Scalable Video Coding (SVC), thereby increasing memory efficiency and reducing the load on caching infrastructure. Software Defined Networking (SDN) coupled with OpenFlow essentially provides the flexibility of forwarding different component flows of the same video file independently. In this paper, we demonstrate how to optimize video delivery by implementing an algorithm that sends layers of different quality via discrete paths. Using this approach we were able to improve the reliability of video playback while reducing the utilization of server resources.

Keywords— MPEG-DASH, H.264, AVC, JSVM, SVEF, SVC, SDN, OpenFlow.

I. INTRODUCTION

i. Statement of the Problem— Adaptive streaming is a process that enables optimum video streaming over a diverse range of devices that are using a variable set of Internet connection speeds. It adjusts the quality of a video based on changing network conditions to ensure the best possible viewer experience.

Adaptive streaming operates over HTTP and creates multiple video files encoded at different quality rates. Although adaptive streaming aims to provide seamless video delivery, it introduces some redundancies (multiple files) in the network, and might encounter problems such as network congestion due to the increase in demand for video services. Thus, it is necessary to develop a highly scalable solution for optimal video delivery that can offer the following functionality:

- Graceful degradation or providing the viewer with an alternative usable version of full functionality in case the less important segments are lost or discarded.
- Adaptive bit-rates at the nodes (including sender) to match channel throughput.
- Robustness to packet loss and bandwidth fluctuation.
- Trade-off between power consumption and quality of video.
- Forward compatibility for older hardware (i.e., the ability of basic hardware to decode a low-resolution subset of the original video).

Advanced Video Coding (AVC) is a block oriented motion compensation based video compression standard. The baseline, main and extended profiles were defined under AVC as independent entities having progressively higher qualities [1]. Adaptive video streaming uses AVC to encode its video segments.
Scalable Video Coding (SVC) is an extension to the H.264 AVC (MPEG-4 AVC) video compression standard, and as the name suggests it provides a very efficient solution to handle multiple streams. SVC encodes video into layers, starting with the “base” layer, which could contain the lowest level of detail in the spatial or temporal sense. Using this layered approach in combination with OpenFlow, which decouples the forwarding plane, different video layers can be pushed through different paths. This can provide a scalable solution for optimal video delivery [2].

Fig.1. Heterogeneous receiving devices streaming video with variable network conditions.

Fig.2. AVC v/s SVC encoding

ii. Research Question— Optimizing video delivery by designing a SVC based model which divides the video into layers of different quality and then routing these layers through discrete paths with the help of OpenFlow.

There are four sub-problems which will need to be answered as part of the research effort, which are as follows:

1. Designing a virtual network to simulate a client server based model

A network topology with multiple paths between the server and client has to be designed so that different video layers can be delivered to the client via discrete paths. For this purpose, a virtualized test bed has to be developed using a network consisting of Open vSwitches deployed on individual virtual machines. Open vSwitch(OVS) is an open-source implementation of a distributed multilayer virtual switch which can be used to simulate a network by creating virtual switches on different virtual machines [3].

2. Utilizing a SVC based framework for video encoding

The ability to support a large user base using a wide range of heterogeneous devices in continually changing network conditions makes video streaming a challenging task. It requires flexible adaptations to deliver the maximum possible video quality and playback continuity to the users. We are trying to provide a more scalable option to optimize video delivery and hence want to use the H.264 SVC extension, instead of traditionally used AVC [2]. SVC encoding is much more complicated than AVC as it involves spatial and temporal scalability in addition to H.264/AVC functionality. Since SVC is not as commonly used in the video streaming industry as AVC, we need to identify a set of SVC encoding and decoding tools. We can then encode a sample video using the SVC encoder and test it on a client with the SVC decoder.

3. Designing an algorithm to determine video delivery paths for different video quality layers

The current Internet provides a best-effort delivery system in which the bandwidth is allocated among all the instantaneous users to the best of its capabilities without any explicit commitment to service quality. As a result, a continuous connection cannot be guaranteed between the client and server especially in peak periods. Software Defined Networking (SDN) provides the capability of selecting the paths between the client and the server based on an arbitrarily defined layer. For example, if all paths of the network between client and server are severely congested, a predetermined layer of minimal quality (the base layer) can be prioritized to ensure a continuous video stream.

4. Implementing the designed algorithm in the virtual network using OpenFlow

Once the different video layers have been obtained, we need to enforce the above designed algorithm such that the network device sends the video packets through discrete paths. This can be done by gaining access to the forwarding plane of network devices using OpenFlow. An OpenFlow switch will send these packets to the controller that manages flow control in an SDN environment. The controller will work according to our algorithm, which will determine video delivery paths for different video quality layers, creating a new flow entry. The switch will now forward
these packets through this new flow thereby improving reliability of the video playback [4].

II. LITERATURE REVIEW

Adaptive video streaming over HTTP is fast becoming the preferred way for video delivery over the best effort Internet. To do away with proprietary streaming solutions, streaming giants like Netflix, HULU and YouTube stream their content using DASH [3]. DASH uses AVC to encode video into temporal segments of different bit-rates. These are then listed in a Media Presentation Description (MPD) or manifest. For any given time during the video's prescribed length, a number of different bit rate options are available. The client downloads and reads the MPD to get important information, such as the content locations, segment encoding, minimum and maximum bandwidth and accessibility features like closed captioning and content restrictions (DRM). The client selects an appropriate segment and begins streaming the content through a series of HTTP requests. The server creates and encodes each segment on demand for each request, all from the same source [5]. Hence, AVC requires a lot of storage space on the server and the segments choose the shortest path available [6].

In SVC, the media stream is comprised of individual complimentary layers. First is the base layer that provides the lowest usable resolution and/or frame rate that any of the SVC compatible endpoints should be able to display. Better quality alternatives can be provided by multiple enhancement layers which can be applied on top of the base layer. These individual layers are additive (i.e., the unnecessary duplication of information between the layers is eliminated) [7]. In SVC, the client adapts to bandwidth fluctuations by dropping one or more enhancement layers.

Additionally, the utilization of SVC results in space saving in terms of server storage. As shown in Fig. 3, as the number of representations (bit-rates) supported by the client increases the storage requirements of AVC increase exponentially. On the other hand, the storage requirements for SVC remain steady and do not increase as quickly as AVC.

A standard networking device can be divided into three planes of operation, the management plane, the data plane, and the control plane. The management plane handles functions such as firmware updates, network management and configuration management via the command line interface. Major functions such as packet processing, routing and forwarding are handled by the control plane and the data plane (also known as the forwarding plane). As shown in Fig. 4, the control plane is responsible for building up the routing information base using the Address Resolution Protocol (ARP). The forwarding plane uses this routing information to build up a forwarding information base and then forwards the packets out to different output ports. SDN provides a dynamic, manageable and highly programmable network architecture by decoupling the control plane and the forwarding plane in an SDN environment. It provides a communication interface which can be used to create flows on the control plane and send them to the forwarding plane.

In order to determine the solution to each sub-problem, the research methodology which will be employed in each case is as follows:

1. To simulate a client server model, we have assembled a test bed in a virtualized environment using the VMware hypervisor. We are using CentOS Linux installs for the VMs. The network between the server
and client is made up of Open vSwitches, which are open-source virtual software based switches. The network topology set-up is shown in Fig. 5. To avoid duplication of packets, every link has been set-up using individual isolated VMWare vSwitches. The isolation of the VSwitches is required as the data strictly has to be passed through an OVS for processing.

The security mode on the ports of the VSwitches is configured to allow promiscuous delivery of packets such that the links can operate as physical links between two devices without having to perform Layer 2 look-up. We opted to establish the network topology using OVSwitches built on individual machines, instead of using a network simulator like Mininet. This has an advantage of avoiding repetitive initiation of server and client setup every time the Mininet topology is restarted. The OVSwitches also provide more granular control over the flow mechanism and interface configurations.

For the OpenFlow controller, we have used a VM running an instance of POX. All the OVSwitches are connected to the controller via an isolated out-of-band vSwitch network so that the video traffic is not affected.

In order to set-up a scalable video coding based framework, we have decided to make use of the Joint Scalable Video Model (JSVM) library for encoding and decoding of the SVC video [10]. To study the delivery parameters, we have opted to use the Scalable Video-streaming Evaluation Framework (SVEF). SVC bit-streams consist of one or more Network Abstraction Layer Units, each having its own NAL unit type [11]. The granularity of adaptation for video delivery can be tuned to the level of one Network Abstraction Layer Unit (NALU) [12]. The SVEF implementation provides a simple procedure of lining the frame number of the NALU and the scalability parameters: LID (layer ID), TID (temporal ID), and QID (quality ID). The LID is used to signal the dependency layer of the sub-stream in the SVC file. This implementation accommodates an RTP header to accommodate these fields in the packet sent out to the client.

For testing purposes, we use a raw video file in the YUV format. The video is encoded using JSVM H264 AVC EncoderLibTestStatic executable. The executable takes a comprehensive configuration file as input, which specifies the input files, the input frame resolution and the input frame rate. It can specify the number of layers of the SVC and the corresponding configuration files for each layer. For our research, we are using SVC videos with temporal scalability and spatial scalability.

In the case of spatial scalability, both layers of the video have similar frame rates. However, layer 0 is encoded at Quarter Common Intermediate Format (QCIF) resolution (176 x 144) resolution, while layer 1 is encoded at Common Intermediate Format (CIF) resolution (352 x 288). In case of temporal scalability, the layer 0 is encoded at half the original frame rate, while layer 1 is encoded at the original rate. This generates a .264 file with playback defaulting to layer 0 characteristics.

To generate a reference trace, the SVC video is decoded at the server with the raw video as input. The video is then passed through the Bit Stream Extractor Static executable to generate a NALU trace file which specifies the sub-streams of the video. Then, we use the SVEF evaluation framework to generate NALU files and delivery of the video. The output from the decoder and the Bit Stream Extractor is used as input to the SVEF F-N Stamp tool, which generates a trace file similar to the output of the Bit Stream Extractor. Additionally, the frame number information is included as well.

Then the SVEF Streamer tool is used to deliver the video to the client identified by a specified IP address and UDP port combination at a certain frame rate. The streamer checks the NALU of the trace file and extracts the corresponding NALU from the video. This is then encapsulated in a RTP packet which has the relevant LID, QID and TID information. The frame format is given in Fig. 6.

![Fig.5. Network Topology](image)

![Fig.6. RTP Header format](image)
receiver. Any corrupted frames or frames with incomplete dependencies are discarded. This generates a trace that can be compared to the original NALU list to generate a filtered .264 file by using the Bit Stream Extractor. The filtered .264 is then passed through the decoder to generate a raw YUV file at the client. The process is outlined in the Fig. 7. The SVC decoding process at the receiver takes place using the MPlayer and the YUV streaming player.

Fig. 7. SVC Encoding/Decoding Process

3. To test the layered scalability of the SVC video, we developed an algorithm with three delivery processes.

a. Normal delivery process

For normal delivery we have implemented a layer 2 learning network between the server and client. Communication happens as it would in a physical switched network with the devices learning about each other through layer 2 ARP (Address Resolution Protocol). We also use the openflow discovery and the openflow spanning-tree components on the POX controller to ensure that there are no broadcast storms as this is a looped topology. Bi-directional flows are installed by the controller in the OVSswitches by inspecting the packets passed to the controller. Once the flows are established, communication occurs in the data plane itself without any control plane intervention.

b. Layer path splitting process

For the path splitting process, we use a layer 2 learning network with some constraints. Bi-directional flows are implemented on all switches except the one connected to the server. The flow between the client and server is unidirectional. As a result, all the packets from the server to the client are sent to the controller for processing. This is done since the OVSswitches cannot implement flows based on the RTP information.

The native implementation in POX does not support the RTP header. So we have created our own packet format for RTP and registered the same in the POX packet library. Data unpacking is performed on the payload provided by the lower layer (i.e., UDP in this case). RTP header information is obtained by making use of the “struct” library in Python. This library allows us to extract characters from the payload. In the header implementation, basic validity checks are performed by checking if the data passed over from lower layers exceeds the minimum RTP header length and also by comparing actual raw data length and the total size field value in the header.

The packet is de-encapsulated at the controller. The IP addresses are checked to confirm server to client communication. The protocol ID is checked to confirm if the packet is UDP, if so, the destination port is checked to ensure that the next header is an RTP header. To implement this, the native implementation of UDP in the packet library of POX had to be changed. Additionally, the length check-sum of UDP had to be bypassed to avoid the length error problem encountered in case of fragmentation.

Once the RTP header is parsed, the layer ID is retrieved. Layer 0 information is sent through the primary path, while information for any other layers is sent through a secondary path. Any packets that cannot be parsed as RTP are sent through the secondary path. This is to ensure that layer 0 travels through a path which has only layer 0 video traffic thereby improving the reliability of playback at the client. Optionally, we can send other network traffic via the primary path.

c. Layer dropping process

For the layer dropping process, we use a network setup similar to the layer path splitting process.
However, any video information for layers greater than layer 0 is dropped, essentially allowing only the layer 0 information to be forwarded to the client.

The forwarding algorithm is outlined in the Fig. 8.

![Forwarding Algorithm Diagram]

4. Implementing the designed algorithm on the virtual network leads to issues related to path MTU (Maximum Transmission Unit) and fragmentation. The test-bed VMs utilize the standard MTU size of 1500 bytes on the OVS and server-client interfaces. The NALUs generated at the server often exceed the MTU size. This causes problems when the packet is processed at the OpenFlow controller. The POX controller learning code does not handle fragmentation; hence we have to handle such packets separately. In this case, the first fragmented bit is identified by the checking the MF (More Fragment) bit and if the Fragment offset is set to zero. The fragment information is then stored in a key value pair with the IPv4 ID as the key and the layer ID as the value. Subsequent fragments are identified by checking if the MF bit is set and the fragment offset is greater than zero or if the MF bit is not set and the fragment offset is greater than zero. The ID of the fragmented packet is retrieved from the dictionary and the packet is processed as per the layer ID. This bypasses the de-encapsulation process. If the fragment ID is not found in memory, the packet is forwarded using the default path. Using this method, the designed algorithm is implemented in the virtual network. Now, we use a network congestion tool known as DummyNet to introduce congestion to the network, so that we can test the video playback under different conditions.

IV. RESEARCH RESULTS

Using the techniques described to solve each sub-problem in the above section, the following results were obtained:

1. **Designing a virtual network to simulate a client server based model**

   Our objective was to set up a basic client-server model that comes close to replicating an actual video delivery network environment, which we achieved using a virtualized network using Open vSwitches on virtual machines. To ensure end-to-end connectivity between the client and server, we used the layer 2 Address Resolution Protocol (ARP) mechanism to allow the switches on the network to learn about each other’s physical addresses. We performed a series of ping operations on the server, client and intermediate switches to ensure full connectivity between the client, server and switches.

2. **Utilizing a SVC based framework for video encoding**

   The SVC based framework was designed by using various scripts in the JSVM reference software library. We used some of these scripts at the server and client side. Encoder Lib TestStatic, BitStream ExtractorStatic, Decoder Lib TestStatic to name a few. We also used the SVEF to introduce a NALU and scalability parameters such as Layer ID that denote the layer number of the sub-stream in the SVC file. Using the above steps a source file in the YUV format was successfully encoded and SVC files were obtained at the server.
3. Designing an algorithm to determine video delivery paths for different video quality layers and implementing it in the virtual OpenFlow network

To send different layers via discrete paths, we initially created our own packet format for RTP. The RTP header was parsed to retrieve the LID of the SVC encoded file. The information for layer 0 was sent through the primary path, whereas the other layers were sent through the secondary path. At the server side, the relevant NALU information was retrieved from the RTP packet and sent via the path corresponding to the LID information. The SVEF Receiver tool was listening to the corresponding UDP port, to generate an output file and a trace file, from which the original raw YUV file was obtained.

The encoded SVC video was now sent from the server to the client using the RTP header. This video has two layers which are encoded temporally; layer 0 is encoded at half the frame rate of the original video (25 fps in our case) and layer 1 is encoded at 50 fps. When both layers are sent via different paths under normal conditions, the output at the client is as shown in Fig. 9. The client uses the MPlayer to play the SVC encoded files. Both the layers of video information were delivered to the client without corruption or excessive delay. The source file was 155MB in size and the total number of frames is 1057. The decoded output at the client is 153MB indicating that there is not much of a difference since the playback at the client involves both the layers.

Now, we use DummyNet to congest the secondary path in our network such that whole of layer 1 can’t reach the client buffer on time and has to be discarded. In this case, the client is still able to play the video but it involves only layer 0 which is encoded at a frame rate of 25 fps and has a total of 133 frames. The size of this file is just 19.2 MB and the video quality is degraded as shown in Fig. 10.

In the last two conditions, we encoded the video based on frame rate, thereby providing temporal scalability. Now, we encode the source video spatially. In this case, both the layers have the same frame rate but they have different resolutions. The original video has a resolution of 352 x 288, layer 0 is encoded in 176 x 144 and layer 1 provides 352 x 288 as shown in figure 11.

Fig. 10. Layer 0 Delivered, Number of frames reduced

Fig. 11. Original Resolution (352 x 288) v/s Only Layer 0 (172 x 144)

Fig. 12 shows a comparison between the original video and only layer 0 played at the client. On congesting the secondary path by running a continuous ping (other traffic) and by dropping some packets using DummyNet, the client receives layer 1 out of its buffer. As a result, it plays only layer 0 which has a size of 38MB and is visually degraded as seen below. In this case, we test for continuous playback by shutting down
the layer 1 link and observing that the client still plays the video as it receives layer 0 through the primary path.

![Image](image.png)

Fig.12. Visible Degradation with lower resolution in Layer 0

V. DISCUSSION OF RESULTS

We were able to simulate a client-server based framework for video streaming using SVC and delivered different quality video layers through different paths as determined by our algorithm. By eliminating redundant information from the video data stored at the server, we are able to save server space and improve caching efficiency. When a test file of 155MB is encoded spatially, the layer 0 file has a resolution of 176 x 144 and size of 38MB and the layer 1 file has a resolution of 352 x 288 and size of 117MB. So, the total space used on the server is 155MB. Whereas in the case of AVC, a low resolution file of 38MB is stored along with a separate higher resolution file of 155MB. Thus, to switch from a lower quality video to a relatively higher quality, the client has to download a total of 155 MB in case of SVC, but this value increases to 193 MB while dealing with AVC. In case of temporal scalability, around 19 MB can be saved in terms of storage space on the server. Also, the reliability of continuous playback improves because the base layer and enhancement layers traverse different paths. On encountering congestion in the path through which layer 1 was traversing, the client does not face any interruption while streaming video, as layer 0 is delivered through the uncongested primary path.

In our implementation the adaptive capability of the video can be made more dynamic by reacting to the congestion in the network links. We have implemented a client that listens passively on the specified UDP ports. No messages are sent to the server to signify congestion information or device capability. Implementing the same over a HTTP model will allow the device to send such information and allow the server to adapt accordingly. For example, if the client is a mobile device, the server can choose to transmit only the base or lower layers since the quality will be sufficient for a smaller screen.

As per our implementation, the intelligence lies in the SDN. But there is a certain amount of overhead that comes with packet inspection at the controller. There are also certain reliability issues concerned with single point of failure at the controller. This can be solved in two ways:

1. The OVSwitches can be modified to allow RTP flow recognition. This will allow us to insert flows just once and the switches should be able to transmit at speeds close to line rate with the right hardware. Controller failure will not affect the video delivery if flows remain active.

2. The intelligence can be transferred to the server. The server can make decisions based on client capability or network conditions and mark packets accordingly. Quality of Service can be implemented using existing methods to allow for the reliable delivery of a certain number of layers. However, this will require modifying the application layer function and could cause cross compatibility issues.

VI. CONCLUSIONS AND FUTURE RESEARCH

The current evaluation frameworks are limited in their scope of options for the delivery of SVC video. There are no comprehensive solutions to set up a server client model with relative ease.

VII. REFERENCES


