AN ARCHITECTURAL APPROACH TOWARDS ACHIEVING BANDWIDTH EFFICIENT CONTENT DELIVERY

Dinkar Bhat, Navneeth Kannan, and Wendell Sun
ARRIS

Abstract

Content delivery is a complex problem given the variety of services to be supported. Bandwidth availability and efficiency at high-end homes becomes key as high bitrate applications including 4K, 4K-HDR, and VR video become available. Performance is usually estimated in terms of latency and different types of latency measurement criteria could be considered. In order to achieve high performance across applications, we consider two aspects: modeling of viewership and efficient adaptive bitrate streaming delivery over IP. We delve into modeling viewer behavior using machine learning clustering techniques that use genre and demographics as features. The aim is to create effective content tranches to which bandwidth can be allocated more efficiently. We then examine the problem of adaptive bitrate streaming delivery of the content to the classes of viewers. In particular, we look at advanced encoding features at the video and transport level that can significantly improve user experience, while preserving video quality. We present the paper from an overall architectural perspective with deeper focus into the above described technical aspects.

INTRODUCTION

The number of services being deployed at homes continues to grow at a very rapid pace. With the introduction of bandwidth intensive applications requiring 4K, 4K-HDR, and now VR video, the architecture for content delivery has to change accordingly. Viewers regardless of underlying network complexity expect low latency, fast availability, and glitchless performance.

We take a look at two technical aspects that would significantly benefit content delivery. First, how should viewership in a network be modeled effectively in terms of their viewer profiles? We describe how they can be clustered based on viewing habits and demographic attributes. Once they are clustered, we argue how content sources can be sliced and linked to the clusters. This linking in turn aids in bandwidth allocation for a delivery mechanism like IP multicast. Second, what steps can be taken to deliver content sources (like live TV services) to viewers with minimal latency. We describe a hybrid network architecture that combines IP multicast with HyperText Transfer Protocol (HTTP)-based adaptive-bitrate (ABR) approach to provide low delay services to clients. We introduce a key component called the ABR-aware Packager/HTTP server that links the IP Multicast cloud with the ABR clients requesting content over HTTP.

VIEWER MODELING

Motivation

As the viewer population grows more diverse and fragmented, the type of content being consumed by different groups becomes quite varied and distinct [4]. The implication is that a program set, which could be a list of channels, can be organized into appropriate slices so that they can be targeted effectively. This would be more effective in terms of bandwidth allocation and developing user interfaces like guides [2]. Figure 1 shows an example of how channels of different genre may be classified based on viewer gender and age.

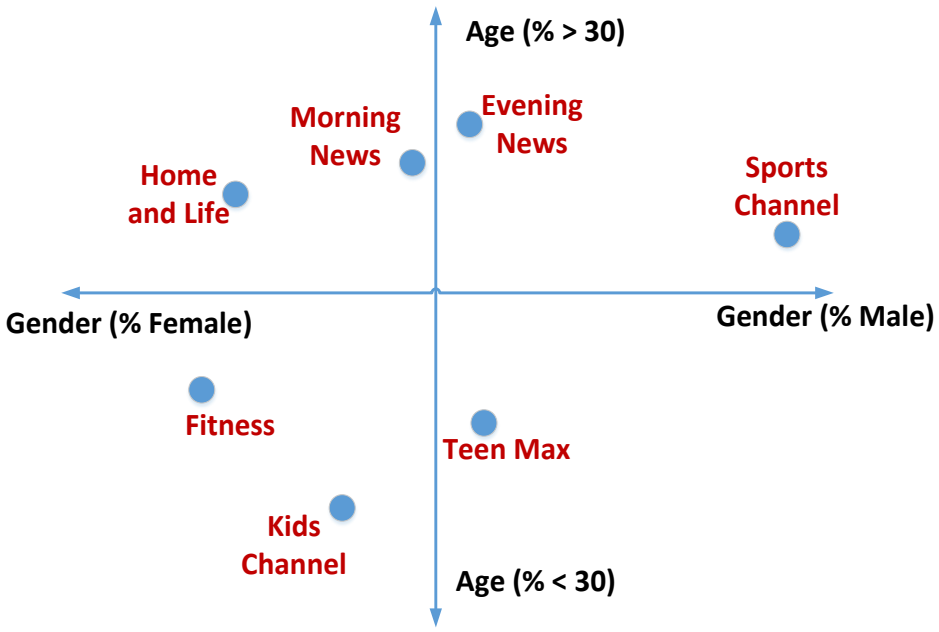

Figure 1: Depicts a sample classification of six different channels based on gender and age

Targeted advertising, which has been the topic of significant technological research due to its obvious commercial importance, is achieved by matching viewer profiles with a set of ads, in order to obtain addressable sets of viewers. Viewer profile attributes including age, buying habits, and income range are obtained from third-party market researchers. In addition, some approaches combine demographic profiles with viewing habits to target ads in specific channels or shows at specific times.

In this paper, we look at viewership modeling from a different perspective, namely the aim is to group viewers into clusters based on viewing habits and demographics attributes, then to use the resulting clusters to create content slices or *tranches.* Once the tranches have been created, they can be allocated appropriately to bandwidth groups (like IP Multicast groups) for transmission (see Figure 2). Note that the tranches can be modified dynamically based on temporal viewing characteristics.

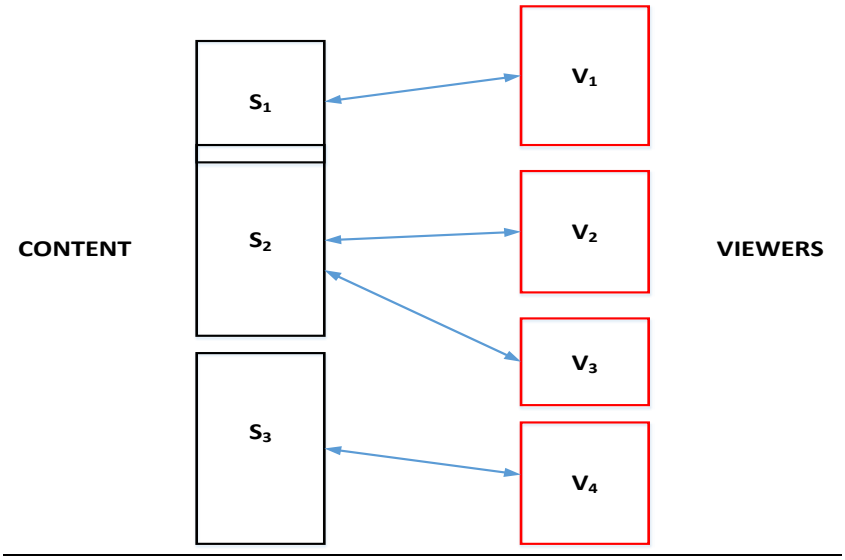

Figure 2: Illustrates a content set divided into tranches and viewer clusters assigned to the tranches

Machine Learning Approaches

Machine learning techniques applied to Big Data now offers significant opportunities into examining viewing behavior, which can then be used to provide better user interfaces, efficient bandwidth management, towards better user experience. In this section, we look at approaches to clustering of viewers based on demographic attributes and viewing behavior.

In general, the aim of clustering algorithms is to find groups within a data set, or clusters that are similar in some defined sense. Clustering algorithms fall in the domain of unsupervised learning techniques because the clusters that may be identified are not known a priori. There is no a priori knowledge of what kind of clusters are present in the data. Hence, a key input to any clustering algorithm is the number of clusters to be identified and often a lot of experimentation is required before an optimal value is used. Figure 3 shows a schematic of how clustering works in general.

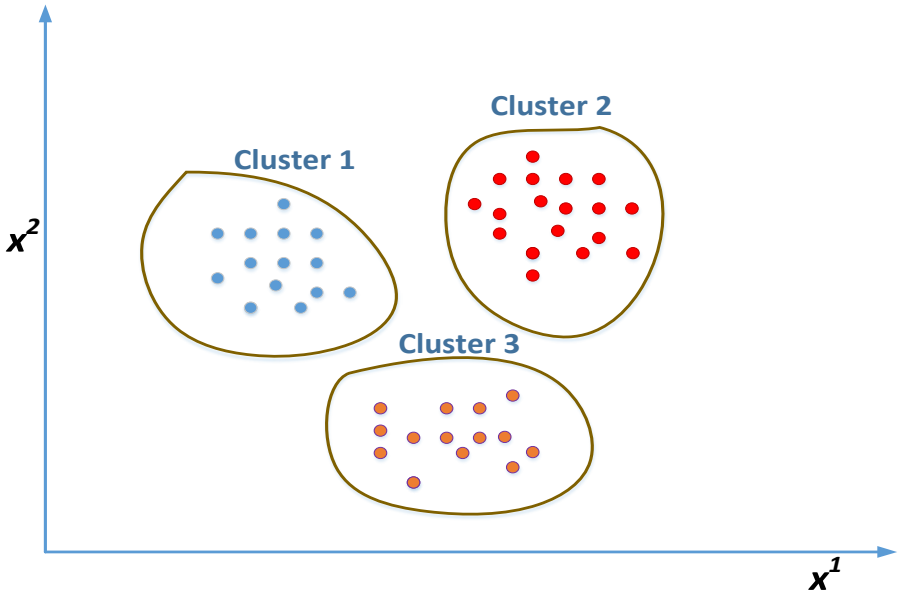

Figure 3: Depicts the possible output of a clustering algorithm (like k-means). The input to the algorithm is the set of data points, each represented by two attributes. Input also includes the number of clusters to be identified (k=3).

We briefly describe a popular clustering approach that could be used in our application, namely, the *k*-means algorithm. Formally, let the set of n data points in our data set be $D = \{\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_3}, .., \boldsymbol{x_n}\}$, where $\boldsymbol{x_i} = (x_i^1, x_i^2, \ldots, x_i^r)$, $i = 1, .., n$ is a vector with r attributes. The *k*-means algorithm partitions the data set D into k clusters, given k. Each cluster has a cluster center, called a *centroid*. The algorithm works as follows: Randomly choose k data points as seeds to be the initial centroids. Repeat the following steps till convergence:

1) Assign each data point $\boldsymbol{x_i}$ to the closest centroid $\boldsymbol{m_j}$ of cluster j, using a measure which is usually the Euclidean distance.

2) Re-compute the centroids using the current cluster memberships.

3) If the centroids have not changed as per convergence criterion, break and output the set of $N=k$ clusters.

During each iteration described above, the centroid $\boldsymbol{m_j}$ of cluster j containing C_j data points is computed as $\boldsymbol{m_j} = \frac{1}{|C_j|} \sum_{\boldsymbol{x_k} \in C_j} \boldsymbol{x_k}$. Selection of the convergence criterion is very important to generate the right set of clusters. The *k*-means algorithm was developed assuming data sets were small, but with large amounts of data its complexity increases prohibitively. Thus, it has seen many novel changes to adapt it to larger data sets (for example [3]).

In our application, each $\boldsymbol{x_i} = (x_i^1, x_i^2, \ldots, x_i^r)$ denotes an instance of viewing. The components of the viewing vector could constitute both demographic variables like age, gender, income, and viewing variables like content genre components and time of day [2]. To prevent too much noise in the data, only instances of viewing where duration was greater than a threshold could be included. k would denote the number of viewer clusters to be output by the algorithm.

To create the content tranches and then match them with the viewer clusters, a simple

approach could be adopted as follows. Let's say the content set of *s* units is represented by

$E = \{\boldsymbol{p_1}, \boldsymbol{p_2}, \ldots \boldsymbol{p_s}\}$ where $\boldsymbol{p_j} = (p_j^1, p_j^2 \ldots, p_j^l), l \leq r,\ j = 1, .., s$ is a vector of size l that contains the same genre components used in viewer clustering. Cluster the content units using the *k*-means algorithm into a set of tranches of size *M*. Now link each viewer cluster (*N*) to one or more tranches (*M*) by comparing centroids using the Euclidean measure.

Bandwidth Allocation

Our approach of dividing content into tranches and linking them to viewer clusters fits well into an IP multicast delivery model [3]. IP multicast reduces traffic in a delivery network by simultaneously delivering a content stream to many recipients. Multicast routing establishes a tree that is rooted at the source with the receivers as the leaves. As opposed to unicast delivery, data in multicast is not copied at the source, but is copied inside the network at branch points of the multicast distribution tree. Thus, only a single copy of data is sent over links that lead to multiple receivers resulting in bandwidth gains. Given that several viewers in a cluster are likely to request content on the tranche they are linked to, the bandwidth savings can be quite significant.

The content tranches are assigned unique multicast addresses. Each receiver in a viewer cluster listens on multicast addresses assigned to the content tranches it has been linked to. Our model, in which several sources transmit on an IP multicast address, is more optimal for an IGMP v3 enabled network where source specific multicast (SSM) is available. In an SSM-enhanced network receivers can signal their intention to join a specific source within a multicast group, unlike a traditional network (i.e. before IGMP v3) where receivers will get traffic from all sources sending on a multicast address. Thus, in an older multicast network there could be unwanted network traffic and it is then up to the receivers to filter data.

Note that although we have assigned viewer clusters to content tranches, nothing prevents receivers from requesting content from other content tranches they are not linked to. This can be done through unicast. We expect that by modeling using content tranches and clusters, such unicast delivery will be more limited. However, the content guides on receivers would list all available content in an intuitive fashion [2] and it would be transparent to the viewer as to how it is delivered. In the next section, we describe a hybrid network architecture combining IP multicast with HTTP-based adaptive bitrate streaming.

ADAPTIVE BITRATE DELIVERY

Common Practice and Problems with Live TV Service

Internet-based video streaming has a long history. For example, IPTV services have been offered by telco operators to compete with cable operators for more than a decade. IPTV services provide subscribers with similar TV viewing experience, such as fast channel change time and low end to end transport latency, as traditional broadcast or linear pay-TV service offers. It uses IP multicast as primary protocol, which provides true data streaming, minimizes delivery latency, and supports content sharing among multiple clients.

However, IPTV services require guaranteed bitrate match bandwidth for smooth service delivering. As a result, an IPTV service is only provided in a managed IP network. Over-the-Top (OTT) services are relatively new to TV service. OTT uses HyperText Transfer Protocol (HTTP) protocol with introduction of multiple bitrate video

source, adaptive bit selection and seamless bitrate switching, which provide transport resiliency to network bandwidth changes, plus the usage of popularly deployed HTTP protocol. Consequently, OTT services are attractive and may replace IPTV as the primary video service in Internet-based video service delivery. However, the HTTP-based transport has major shortcomings, especially for live or linear video delivery.

The HTTP protocol is designed for transaction based file transfer. It may be perfect for a Video-on-Demand (VOD) type of video service, especially with progressive file download, where a user can watch the downloaded video even before the file download is complete. However, it is not quite suitable for live streaming. The current HTTP Adaptive Bitrate (ABR) streaming uses small segments to compromise HTTP file transfer request. Normally there is a manifest file that describes availability and URL of media segments. Media transport and playback process is initiated by HTTP ABR client via sending a HTTP request to get the manifest file, then the client selects a media segment in an adequate bitrate and starts download it. The client continues the media segment request one after another either in the same bitrate or in a different bitrate based on network bandwidth condition. Each media segment delivery is a HTTP transaction.

In the case of a live TV service, the manifest file will have to be updated to catch up with newly available segments while the media source moves forward in timeline. The client also needs to get the updated manifest file to view and select newly available media segments. When an HTTP ABR client asks for media segment, the HTTP server does not respond until the media segment is available. From a media transport perspective, the delay of live media transport is at least the length of segment besides other delays caused by media processing, such as encoding. The bigger the segment, the longer the delay latency is.

Taking a normal HTTP Live Streaming segment, for example, usually its length is 10 seconds, then the media transport end to end delay is about 15-20 seconds or more. If this forces us to create smaller media segment such as 1 or 2 seconds, to reduce the delay, it causes two other issues. For the reason of seamless switching between bitrate streams, the media segment is bordered by an IDR frame, which requires full decodable I frame. The smaller the media segment is, the more I frames are inserted. The more I frames exist, the more coding resource is required, thus the worse video encoding efficiency is achieved.

In addition, each segment delivery corresponds one HTTP get/reply transaction. The smaller the segment is, the more HTTP protocol traffic is in network.

This type of segmentation delay does not exist in an IPTV service that uses IP multicast technology. In this paper, we will propose a hybrid network architecture, which combines IP multicast in backbone network and HTTP in edge access network. With usage of HTTP Chunked Transfer Encoding [1], it minimizes the delivery latency for live/linear video service without sacrificing video encoding efficiency.

Hybrid Network Architecture

IP multicast has been used in IPTV services and is proven to be capable of delivering low delay media to TV viewers. Why cannot we continue to leverage its benefit and use it for ABR OTT service? At the same time, cannot we also preserve the popular usage and attractiveness of HTTP protocol, especially when it interacts with ABR client?

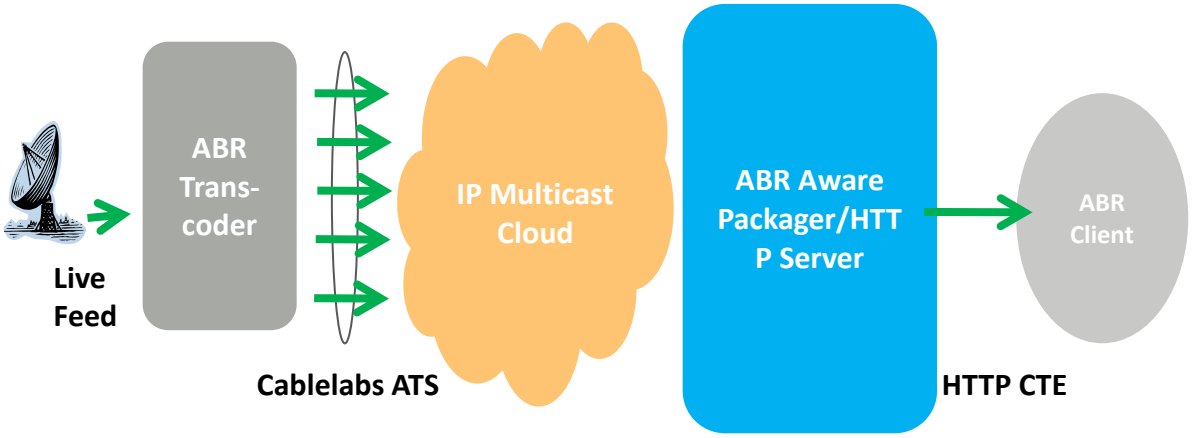

Figure 4: IP Multicast + HTTP Chunked Transfer Encoding

Figure 4 shows the hybrid network architecture, in which IP multicast is used in backbone network and HTTP is used in client oriented access network.

The live source is prepared by an ABR transcoder to create multiple bitrate streams per ABR OTT service profile. These streams are MPEG-2 Transport Stream (TS), just like legacy IPTV media streams carried in IP multicast network. The difference between the IPTV case and ABR OTT scenario is the number of streams. The IPTV has one single stream while the ABR OTT has a set of streams.

The IP multicast cloud is a backbone transport network. In general, the backbone network is well designed and engineered to meet media transport requirements. For example, the IP multicast cloud should be able to deliver all ABR streams in the case each of them is requested by a client. At the minimum, it should guarantee delivery of the stream in the lowest bitrate to keep smooth playback in client.

A new component, the ABR Aware Packager and HTTP Server, is introduced in Figure 4. It is different from a normal ABR HTTP server, which receives media segments from an ABR packager and simply serves ABR client via HTTP protocol. Instead, it is a Just In Time Packager (JITP) that can accept MPEG-2 TS carried in IP streams and generate client requested media segment **in real-time** when it responds to the client request for a media segment.

Similar to a normal ABR HTTP server, the JITP/HTTP server composes and publishes an ABR manifest file. For live/linear TV service, it just needs to tell what bit-rate/resolution streams are available with a virtual segment URL. When it receives a media segment request from ABR client, it checks if the requested bitrate stream is available or not. If the bitrate stream does not exist, it will join the IP multicast group of that bitrate stream. After the server receives the IP multicast stream, it starts to create the requested segment and prepare to serve the ABR client.

As we discussed in the previous section, if the JITP/HTTP server does not send the client requested media segment until the segment is ready, it still bears the extra segmentation delay to the media transportation. What we expect is to send the media segment incrementally in small chunks while the JITP/HTTP server receives and prepares it.

HTTP Chunked Transfer Encoding

The HTTP Chunked Transfer Encoding (CTE) is defined by HTTP/1.1: Message Syntax and Routing [1]. It is a data transfer mechanism in HTTP 1.1, in which data is sent in a series of "chunks". It uses the Transfer-Encoding header, instead of the Content-Length header. The HTTP sender does not need to wait for the total size of content and can start sending data as small chunk with any amount available while still receiving the content. When the chunk is sent, its size is indicated in the Transfer-Encoding header. If the chunk length is set to zero, then it is the end of content.

This is what exactly we want, isn't it? When the JITP/HTTP server receives media segment request from ABR client and the corresponding IP multicast stream is available, it starts the segmentation process, but it can start sending the media chunks in meaningful small size, such as small as one video frame, while it still receives the IP multicast stream. At the end of segment, it just needs to send a zero sized chunk to finish the segment transfer. And then the ABR client repeats sending segment request one after another, the JITP/HTTP server continues serving it with incrementally delivered chunks of segment. In this way, the delay caused by media segmentation is eliminated.

Combining with the usage of IP multicast in backbone network, this hybrid network approach, together with the help of HTTP CTE, can reduce the end to end transport latency to minimum and makes ABR OTT service be comparable with IPTV service.

CONCLUSION

We described an approach that combines viewer modeling with a hybrid network architecture to delivery content to ABR clients. The aim was to allocate network bandwidth effectively based on viewer clusters and then to deliver the content using a combination of IP multicast and HTTP-based ABR streaming to clients. Combining technologies is likely to provide much better performance with the advent of high bitrate services.

1. IETF RFC7230, Section 4.1, "Chunked Transfer Coding", http://tools.ietf.org/html/rfc7230#section-4.1

2. Bhat D., Kannan N., "Adaptive Television User Interface Using Machine Learning Concepts", IBC 2014, September 2014.

3. Legout, A., Nonnenmacher, J. and Biersack, E. W. , "Bandwidth Allocation Policies for Unicast and Multicast Flows", Infocomm 99, March 1999.

4. Cohen, J. "Television viewing preferences: Programs, schedules, and the structure of viewing choices made by Israeli adults", Journal of Broadcasting & Electronic Media, June 2002 204-221.