Analyzing user video download behavior with clustering and Hidden Markov Models

Frederic Beister
Fachgebiet Rechnernetze
Universität Paderborn
Warburger Str. 100, 33100 Paderborn
Email: frederic.beister@uni-paderborn.de

***Abstract*—A large portion of future network traffic in mobile networks will be due to video streaming. Anticipative networking in such a scenario benefits from knowledge about the behavior of video streaming clients in order to manage resources. In streaming technologies such as MPEG DASH and HTTP Live Streaming (HLS), video is delivered in segments which need to be explicitly requested by the client. Depending on the client and its situation, this generates patterns of segment downloads which can be described by their inter-download times. I use Hidden Markov Models and clustering techniques in order to build a model for these inter-download times with which they can be predicted.**

I. Introduction

The goal of anticipatory networking in a mobile network is to allow the basestations and backhaul network to adapt to future load situations before they arise. This means that for low load situations some basestations can be disabled to save energy while more basestations need to be active in high load situation. To reach this adaptation, two problems have to be solved. On the one hand, once a load situation has been predicted, an anticipatory mobile network management needs to decide on what components to activate or deactivate and how to configure them. On the other hand, the future load situations have to be predicted from past and current data.

Current forecasts predict that video streaming will make up a large portion of mobile traffic in the next years [1][2]. Also, the overall traffic demand on basestations is predicted to rise drastically. If the upcoming traffic demand of video streaming sessions as well as the future locations of the corresponding user were predictable, this would mean that a large part of the future load situation was predictable. This paper deals with predicting the traffic demand of video streaming sessions through prediction of segment inter-download times in segmented video streaming such as MPEG DASH [3] and HLS [4].

I used real-world traces from a mobile network which consisted of HTTP requests sent over the network during 41 days between December 2011 and January 2012. In order to cope with the raw data, which was more than 7 terabytes, I used the BigData framework Apache Hadoop to extract the inter-download times of segmented video streaming happening in that network during that time. I used all entries which, according to the MIME-type, referenced video files and whose filename ended in `.ts`. I then assumed that all `.ts` files residing in one folder belonged to one stream. For the remainder of this paper, the term *viewing* describes a list of inter-download times for one pair of user and stream. The data yielded 3.9 million viewings.

II. Preliminary analyses

In order to understand the data at hand, I did some empirical analyses of it via histograms, boxplots and statistical metrics such as average and standard deviation. I saw some very large inter-download times which seemed to result from the fact that a user watched the same stream multiple during the month. I decided to introduce a cut-off at 3600 seconds (1 hour). This means that each inter-download time larger than the cut-off value resulted in the viewing being split. Figure 1 shows that the choice of 3600 seconds leaves the largest part of the viewings untouched as most inter-download times are even smaller than 1000 seconds.

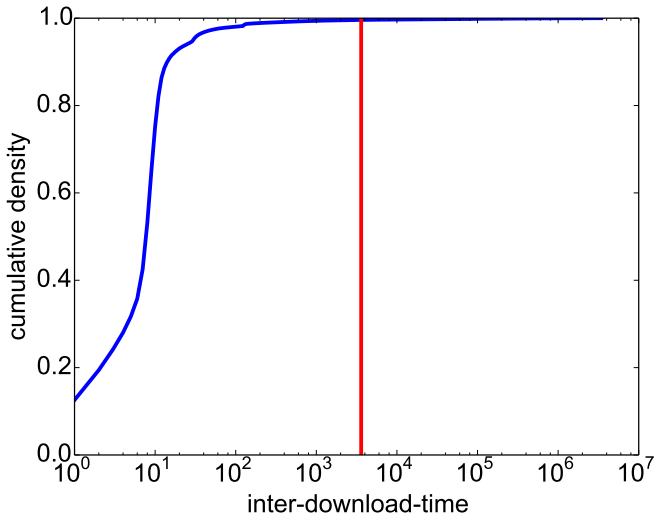

Fig. 1. Cumulative density over all inter-download times, 3600 second cut-off marked by a vertical line

After eliminating large inter-download times, some viewings showed patterns which were clearly explainable with client behavior wheras other viewings had no obvious patterns. A very common pattern was the pre-buffering pattern where the client downloads some segments in advance and then only downloads further segments once a segment has been played from the buffer. A typical pattern of inter-download times would be `0,0,0,5,5,5,5,5,5,5,...` where the client pre-buffers three segments (most probably with a length

of 5 seconds) and then loads segments once a segment has been removed. Another possible pattern with pre-buffering is `0,0,0,10,0,0,10,0,0,...` where the client loads segments once only one segment is left in the buffer.

As the data was from a real-world system, it was very noisy because behind the inter-download times stood a client which was possibly moving in and out of coverage. The inter-download times thus depend on three different processes, namely buffering behavior of the client program, video viewing and mobility behavior of the user and channel conditions at the user location. This meant that patterns as the ones above were very rare to find exactly like this. This lead me to the idea that some pre-processing was needed as well as some degree of fuzziness in the analysis.

III. Inter-download intervals

I introduced inter-download intervals in order to cope with the noisyness of the data at hand. I figured that a network management system based on the load prediction generated from the inter-download times would not gain much from knowing the exact seconds of the next segment requests but could function properly with time intervals. Introducing these inter-download intervals (IDI) allowed me to shrink the number of possible values, smoothen the data and thus remove some of the noise. The definition of these intervals were an added parameter to analyse. I analysed two IDI definitions:

- **Equal probability:** All intervals are equally likely to occur. This is calculated from all inter-download times and resulted in small intervals for small values (occuring more often) and larger intervals for larger values.
- **Custom/Scenario-specific:** I defined the intervals such that they reflected the next segment being needed *immediately*, *very soon*, *soon* and *later*. The idea behind this approach is that actual network management systems can define their own intervals depending on the algorithms they implement. This also allows to analyse how the choice of IDI influences the prediction quality.

IV. Modeling with Hidden Markov Models

Hidden Markov Models (HMM) are a versatile tool in unsupervised learning introduced by Baum in 1966 [5]. The only choice I neede to make, apart from the possible emissions which were given by the IDI, was the number of states which became my second parameter. The Baum-Welch Algorithm can then train a randomly initialized HMM to best reflect given emission sequences. Further well-known algorithms such as the Forward-, Backward-, and Viterbi-Algorithm can answer all relevant stochastic questions related to HMMs. The questions I wanted to answer were:

- **Prediction:** Given a partial sequence of inter-download times, when will the user download the next segments (i.e. what is the most probable continuation of the sequence of inter-download times?)
- **Prediction quality:** Given a partial sequence of inter-download times, how probable is the HMM predicting the correct next inter-download times?
- **Model quality:** Given a full sequence of inter-download times, what is the probability that exactly this sequence was generated by the model?
- **Model structure:** Can we deduce clients' download strategies from the resulting model?

A. Prediction and Prediction quality

Once a trained HMM exists, the most probable next emission can be calculated using the Forward-Algorithm. The algorithm can calculate the most probable state the HMM was in when each of the symbols in an emission sequence was emitted. This can be used to calculate the most probable state the HMM was in when the last emission in a known sequence was made. From there we can look at the most probable next states and their emission probabilities to get the most probable next emission.

Prediction quality is a bit more complicated to describe. I introduced the so-called *p-value* in order to describe how well a trained HMM fits to a given emission sequence. The p-value of such a pair is the average probability at which the above mentioned algorithm correctly predicts each of the elements of the sequence given only their predecessors. The average p-value over many sequences is then the *p-metric* of the HMM for these sequences.

Figure 2 shows the p-metric for different number of states and different IDI. Additionally, I looked at how relaxation of valid predictions influenced the outcome. The graphs titled *smaller* and *builtin* allow the prediction to be smaller than the actual value to some, varying, extent. This is relevant because network management systems might not have a problem if the segment is requested later than predicted.

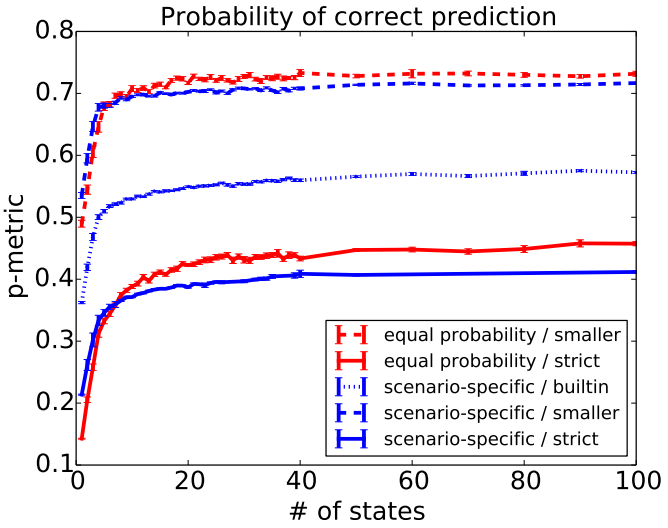

Fig. 2. p-metric for different configurations, confidence intervals at level 0.95

The results show that about 15 states are enough to describe the data as there is no significant increase in prediction quality beyond this point. Also, only around 40% of the predictions are correct which increases to 70% if all smaller predictions are deemed valid (mode *smaller*)

B. Model structure and quality

Looking at the resulting HMMs yielded only small insight into the structure behind the inter-download times. One problem I identified was that many different users and their client's behavior were modeled by a single HMM. For example, when an HMM is trained on the sequences `0,1,0,1` and `0,2,2,0,2,2`, sometimes also sequences like `0,1,0,2,2` became valid and very probable. This to some extent also explains the prediction quality of only 40%.

V. Adding Clustering

In order to overcome the limitations of a single HMM as mentioned above, I looked at possibilities to create multiple HMMs where each HMM represents a certain group of users. As a first step, I needed to identify these groups. This identification needed to be quickly calculatable from the inter-download times available because it needed to be performed online when a user was moving through the network. I used basic metrics on the inter-download times in order to identify clusters. I used the 5- and 95-percentiles as well as the mean, standard deviation, skew and length of the traces. This led to a six-dimensional point for every viewing.

I then used Primary Component Analysis (PCA) in order to prepare the data for clustering and shift the variance to the first dimensions. The first four dimensions after PCA explain more than 95% of the variance. I had to keep an eye on the curse of dimensionality, which states that for an additive increase in dimensions, in order to keep sample density and thus clustering performance high, one needs an exponential increase in sample count. This means that with the 3.9 million samples in six dimensions I have the same density as with 12 samples in one dimension. Using only the first four dimensions would increase this density to 44 samples in one dimension. As I did not know how well the clustering algorithms would perform, I just kept all six dimensions knowing that I could easily increase the density later without sacrificing too much explained variance.

I then applied different clustering algorithms to the data before creating and testing HMMs for each cluster. I used k-Means [6] with varying k from 2 to 40, DBSCAN [7] and OPTICS [8].

Figure 3 shows the resulting p-metrics after the clustering for each of the clusters. DBSCAN returned only one cluster and OPTICS would not finish after running for more than seven days. Some k-Means results have been left out due to them being very similar to the ones shown here. The width of the bars is proportional to the size of the cluster they represent. The reference value is equal to the scenario-specific / builtin value for 15 states in Figure 2 which can be seen as being generated from a clustering with only one cluster.

The graph shows that clustering with the chosen metrics slightly increases the overall prediction quality with growing k. This can be seen by comparing the blue area above the red line to the white area below the red line. If the first is larger than the second, the prediction quality is better than the reference.

VI. Conclusion and Future Work

The prediction of inter-download times allows for interesting applications in anticipatory mobile networks. Using prediction through the HMM approach presented here, network management systems could prepare the network for upcoming load situations. Using the custom inter-download intervals, the management system can get detailed information on what will happen in the future.

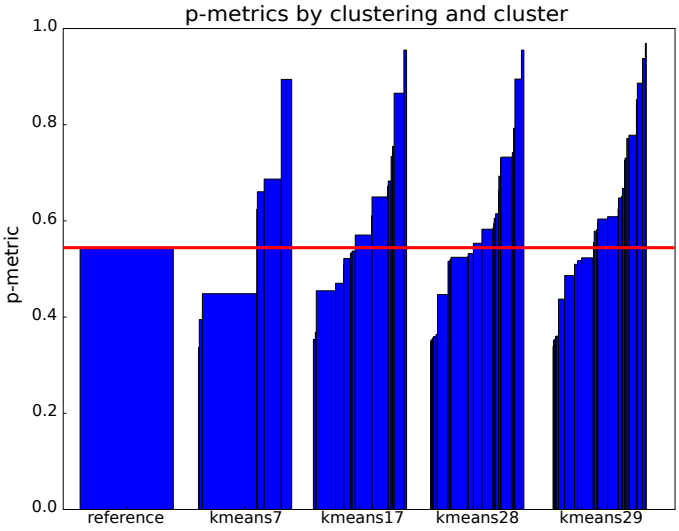

Fig. 3. p-metric for different clusterings

Without clustering, the HMM approach yields a prediction quality of only $40 - 60\%$ which can be improved by intelligently clustering viewings before creating HMMs. The results shown here only cover one set of metrics. Finding a better set of metrics to cluster on remains as future work. Furthermore, using other approaches such as clustering via Apache Hadoop, might allow for more complex clustering algorithms to be used.

Some aspects such as user location, user behavior and channel conditions could not be analysed because the trace did not contain information about the client location or channel quality. Enhancing the clustering by adding new metrics such as user type or location could improve the prediction quality without changing the basic idea of the process.

Being able to work with such a large amount of real-world data allowed for interesting analyses. For my research, I have selected only a very small part of what is in that trace and still got more than 3.9 million data points.

References

[1] Cisco White Paper, "Cisco visual networking index: Forecast and methodology, 2013-2018," 2014.

[2] J. Erman, A. Gerber, K. K. Ramadrishnan, S. Sen, and O. Spatscheck, "Over the Top Video: The Gorilla in Cellular Networks," in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC '11. New York, NY, USA: ACM, 2011, pp. 127–136. [Online]. Available: http://doi.acm.org/10.1145/2068816.2068829

[3] ISO/IEC 23009-1, "Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats," 2012.

[4] "HTTP Live Streaming," http://tools.ietf.org/html/draft-pantos-http-live-streaming-12, 2013.

[5] L. E. Baum and T. Petrie, "Statistical Inference for Probabilistic Functions of Finite State Markov Chains," *The Annals of Mathematical Statistics*, vol. 37, no. 6, pp. 1554–1563, 12 1966. [Online]. Available: http://dx.doi.org/10.1214/aoms/1177699147

[6] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, Mar 1982.

[7] M. Ester, H. P. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, 1996.

[8] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "OPTICS: Ordering Points To Identify the Clustering Structure," *ACM SIGMOD international conference on Management of data*, 1999.