POPS: a popularity-aware live streaming service

Karine Pires, Sébastien Monnet and Pierre Sens
Sorbonne Universités, UPMC Univ Paris 06, Équipe REGAL, LIP6, F-75005, Paris, France
CNRS, UMR_7606, LIP6, F-75005, Paris, France
Inria, Équipe REGAL, F-75005, Paris, France
Email: firstname.lastname@lip6.fr

***Abstract*—Live streaming has become very popular. Many systems, such as justin.tv, have emerged. They aim to collect user live-streams and serve them to the viewers using broadcasting servers. However, the huge variation in the total number of viewers and the great heterogeneity among streams popularity generally implies over-provisioning, leading to an important resource waste. In this paper, we show that there is a trade-off between the number of servers involved to broadcast the streams and the bandwidth usage among the servers. We also stress the importance to predict streams popularity in order to efficiently place them on the servers. We propose POPS: a live streaming service using popularity predictions to map live-streams on the servers.**

I. Introduction

We are currently witnessing the emergence of two phenomena: the popularization of video capture devices and the explosion of network quality and the number of Internet users. Indeed, nowadays, any laptop, netbook or even cell phone has a camera. Full HD devices have become affordable, including their mobile version. It implies that a large portion of the population has the necessary equipment to create streaming video.

Regarding networks, we have seen in recent years an influx of users and a globally better coverage. In parallel network quality in terms of latency and throughput improved significantly, and this improvement continues, in particular with the arrival of very high speed networks like the optical fiber. In addition, connection charges have become reasonable, and the vast majority of Internet users has unlimited access. The mobile network coverage is extensive and of good quality. The progression rate of the coverage of mobile broadband (3G/4G) suggests that the most part of the population will almost have a permanent access to a network of good quality at a reasonable price.

Combined, these two phenomena, and the tendency of users to expose portions of their lives led to the rise of a new type of system: live video streaming systems. These systems, such as justin.tv [1] or YouTube live [2] offer to users the possibility to broadcast or to watch live video streams. The amount of data to be processed is considerable. For example, every minute, justin.tv platform must absorb more than 30-hour live video stream created by thousands of contributors [3]. Streams must be broadcasted to hundreds of thousands of viewers.

The main difficulty in such systems comes from: (i) the scale, both in terms of video streams and in terms of number of viewers; and (ii) the high popularity variation, among both time, and streams. The problem of live streams diffusion is different from Video on Demand (VoD): it is not possible to replicate video files *in advance* on multiple servers, there is no video file before the start of a stream. It also differs from IPTV: compared to television, the number of different streams is greater and the number of users per stream is extremely variable, with some streams having very few viewers.

It is thus necessary to dynamically adjust the number of streams of broadcasting servers to meet the demand. When a stream becomes very popular, it is possible that the server which distributes it reaches its maximum capacity, in that case it must share the load with another server. This induces an additional cost in terms of bandwidth consumption between servers. To minimize this overhead, it is necessary to smartly map the video streams on servers. In this paper, we propose POPS, a popularity-aware live streaming service. Our approach is based on a study of real traces that we collected on the justin.tv and YouTube live platforms. It allows us to predict the popularity of a video stream. POPS is based on these predictions to dynamically provision broadcasting servers. Our evaluation shows that when a history can be used to estimate the popularity of a stream, POPS can decrease the amount of bandwidth used among servers without using too many servers.

The remainder of this paper is organized as follows. Section II details the context of this study and describes related works. Section III presents our contribution: a placement strategy using popularity predictions. Finally, Section IV describes our evaluation before Section V concludes.

II. Context and positioning

This section details the context of our study and presents related works.

A. Model

We focus on platforms broadcasting user generated live video streams. Users of these services can play both roles: some create video streams, they are called *uploaders*; some watch live streams, they are called *viewers*. At a given time, a user can play either one of the two roles, or both simultaneously. However, in our study, we focus on each role separately. It is also possible for a user to watch multiple streams simultaneously, even if it is not norm.

In our model, illustrated by Figure 1, when a user creates a video stream, he sends it to the platform. The platform consists of a set servers. The number of involved servers may evolve dynamically. Servers can be virtual machines in a *cloud* and be

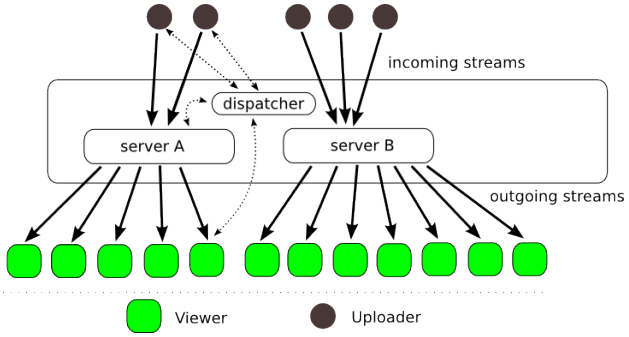


Figure 1: Live stream overview.

provisioned according to the load. The servers can be in a same data center but they can also be geographically distributed, in different data centers or located at the edge of the network. We do not consider limits on the maximum number of servers used by the platform, and we neglect the time it takes to allocate a new server (servers can be pre-allocated, by pool). Each stream is assigned to a server that is responsible for broadcasting it to the set of viewers. A server can be assigned to broadcast multiple streams. It is also possible that a stream is supported by more than one server, when its popularity (number of interested viewers) exceeds its server's capacity.

We consider a *dispatcher* service that could be either centralized or distributed. This service receives requests: (i) to broadcast a new stream, from an uploader; (ii) to subscribe to a stream, from a viewer. This service is responsible for assigning a server to broadcast a new stream, it may either choose an already existing one that has capacity left, or allocate a new one. Then, the uploader sends its stream directly to the assigned server. While receiving a request from a viewer, the dispatcher service identifies the broadcasting server in charge of its diffusion and returns it to the user. The server will then send the stream directly to the viewer.

The design of the dispatcher service architecture is beyond the scope of this paper. We here focus on the study of placement choices that such a service has to do and their consequences on the number of needed servers and on the amount of inter-server bandwidth consumed. The number of clients that can be served by one server depends on the server capacity (in terms of outgoing bandwidth) and on the served streams bit-rate. A server can therefore broadcast to: $\frac{server\ outgoing\ bandwidth}{\sum_{i=1}^{\#viewers} bandwidth\ needed\ for\ viewer_i}$ destinations. A destination can be either a viewer, or another server when a stream has to be served by multiple servers (see Section III). For each stream it serves, a server preserves enough bandwidth to be able to send one copy of the stream to another server in the case new viewers arrive and it does not have the capacity to serve them. We do not consider the incoming bandwidth limitation in this paper: in most systems we observed around two orders of magnitude more viewers than streams. That means that the scalability problem is at the level of the outgoing bandwidth, to serve a high number of viewers. We thus consider that the incoming bandwidth is always sufficient.

The scales are important in our problem. The number of users that these platforms must be able to handle at a given time is of the order of several hundreds of thousands. Since

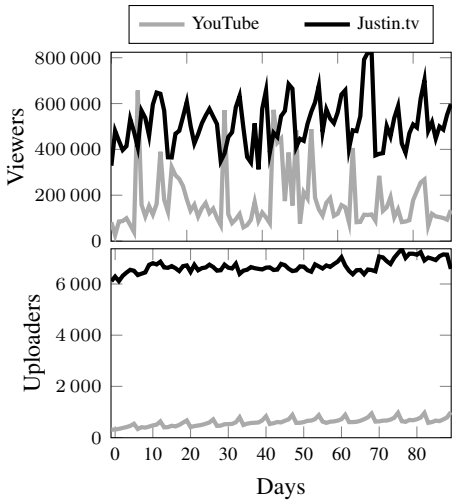


Figure 2: Number of users, viewers on the top and uploaders on the bottom.

the information of each uploader identification is available in the systems we crawled, we were able to compute the total number of uploaders. At the end of the analysis period, over one million different uploaders have been observed on justin.tv and more than hundreds of thousands on YouTube. There are always many video streams (thousands), but also many viewers (hundreds of thousands spread over the different streams). The information collected about viewers is presented in an aggregated way: at any given time, for each stream, the total amount of viewers is available. Therefore, we use and present only the aggregated information, as for example the peak of viewers. Computing the total number of unique viewers would require more precise information, for example each viewer's identification, which is not provided by the platforms. Table I summarizes these values. The distribution of viewers among the streams is very heterogeneous: a few units to thousands and even hundreds of thousands for some of the most popular streams. Figure 2 shows the values during the period from January 6 to April 6 (2014).

	justin.tv	**YouTube**
Number of uploaders (total)	1,068,138	120,097
Number of incoming streams (total)	5,221,208	527,677
Uploaders peak	7,369	994
Viewers peak	823,227	657,087

Table I: Overview of justin.tv and YouTube scale

B. Positioning

We have identified four main categories of related work.

Measurements studies on video content. Many studies focus on the analysis of video traffic. Authors of [4] studied the YouTube traffic generated by mobile devices and compared it to the one generated by standard PCs. They showed that the access patterns remained comparable. In [5], YouTube bandwidth and storage space needs are studied, as in [6] that studies bandwidth from an Internet service provider point of

view. All these studies show the increasing importance of video streams systems and the need to design scalable platforms to broadcast this type of data. However, we did not find any, freely available, large scale trace of user-generated content live streaming platforms. That is why we have run our own collection campaign on the justin.tv and YouTube systems (see Section III).

Peer-to-peer systems for user-generated live video streaming. In POPS, we focus on solutions based on multiple servers to distribute video streams. Other studies adopt a completely decentralized approach. A detailed state of the art is given by [7].

However, the majority of these studies focuses on decentralization and the topology of the peer-to-peer overlay network [8], [9]. It is important to consider an approach based on a set of servers, because it is usually the one chosen by commercial systems [3] and actually works on a large scale, being able to attract a large number of users. There are also solutions using networks such as CDN (*Content Delivery Networks*) [10], [11], tailored for extremely popular streams. In our opinion, this kind of approach is adapted to a small number of extremely popular video streams, like what is observable in the context of IP TV.

IP TV. The study presented in [12] provides an analysis of one of the largest European live video stream provider: Zattoo. In this paper, the authors have information on the architecture of Zattoo from a provider point of view. However, Zattoo is not really a large-scale system: the observed load peaks are an order of magnitude below those we have seen in our data collection. Other works, like [13], study the load of IP TV systems. However, these systems have fewer channels than systems offering user generated contents. The viewers of the systems we target are spread over a large number of video streams.

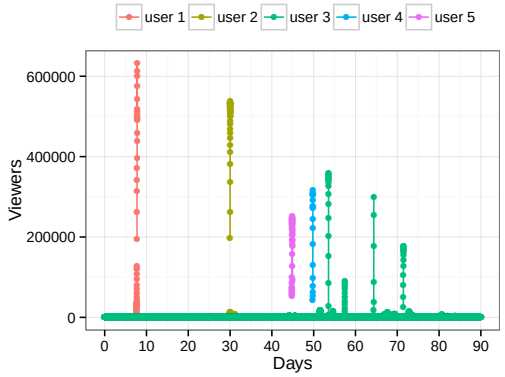
Video on demand delivery systems. Finally, there are research efforts on the data placement for VoD systems. [14] defines an approach that solves a mixed integer programming problem in order to find the best placement for the videos. This work however relies on video migration and caching, both out of the live scope. Another caching strategy focused on energy efficiency is proposed by [15]. The fact that the video streams we consider are *live* changes the problem's input. It is not possible to place video files in advance on servers for instance. Furthermore caching strategies would impose delays that are not desirable on the live scenario. In this scenario there is an important number of viewers watching the same stream at the same *offset*, contrary to what can be observed in VoD systems.

III. Mapping live video streams on broadcasting servers

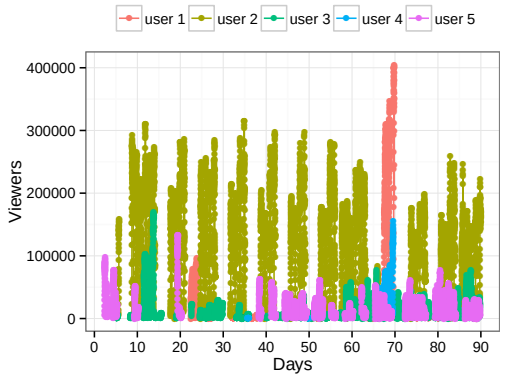
This section describes our contribution. We begin by briefly present the lessons we have learned from the collection of justin.tv [1] and YouTube [2] user traces. We study different placement approaches to map video streams on broadcasting servers. We present our approach based on stream popularity prediction.

A. *Justin.tv and YouTube trace analysis*

Justin.tv and YouTube are user-generated live video streaming platforms. Justin.tv is very popular: cumulative over three months, there have been more than five million streams, generated by more than a million uploaders. Everyday, several hundreds of thousands of viewers are simultaneous using the services. Furthermore, both platforms offers APIs providing the ability to periodically collect streams and users information. All the data we have crawled from justin.tv and YouTube are available on demand.



(a) YouTube



(b) justin.tv

Figure 3: Uploaders sessions over the days of the five users with biggest peak viewers

A complete analysis of the collected data is out of the scope of this paper. The main lessons we have learnt from the data are:

1) among video streams, the popularity is highly heterogeneous, the biggest part of the traffic being generated by the few most popular ones, Figure 3 presents five different users and the heterogeneity between streams;
2) the global system load varies a lot during time, remind Figure 2 that provides the viewers over the

collected days;

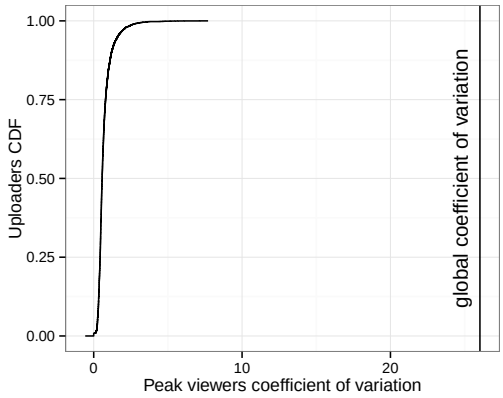
3) there are always many video streams; orders of magnitude higher than the number of channels offered by a classical cable TV, again the comparison of both curves (viewers and uploaders) on Figure 2 illustrates this point;
4) users behavior is predictable: it is possible to observe a night and day oscillation, but also that a stream popularity is highly correlated to the *uploader* that creates it, and it tends to grow with the stream length. That means that, popularity among uploaders is highly heterogeneous, but the streams of a same uploader have a similar peak popularity.

B. Popularity predictability discussion

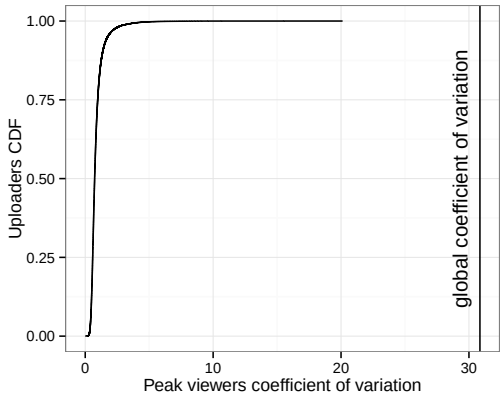
The popularity prediction in live stream services is a key aspect for resource provisioning but it is far from being trivial. We extracted from the data that the users (the uploaders) popularity magnitude is predictable but not the precise popularity (the number of viewers the future streams will gather).

It is not trivial task since among all the uploaders there are extremely different behaviors. To illustrate this, Figure 3 shows the popularity evolution over time for five selected users from both services. We select the five users that reached the highest peak of viewers over the collected period. Notice that, between the two services (YouTube and justin.tv) there is already an important difference: on justin.tv popular uploaders broadcast more frequently than on YouTube. The popularity differences between uploaders of a same service provider are clear. Taking as an example users 2 and 5 on justin.tv, user 2 is about five times more popular than user 5.

However, sessions (i.e., streams) that are made by a same uploader have a tendency to be similar. While taking a closer look at users 2 and 5 of justin.tv, it is easy to identify such trend among each uploader's streams. User 2 popularity stays around 250 000 viewers and user 5 around 80 000. To present such a tendency we calculate the popularity coefficients of variation presented by Figure 3. The coefficient of variation among all streams is represented by a vertical line, while the coefficient of variation for streams of each uploader are presented by the cumulative distribution curve. For simplicity, we only take into account uploaders with more than 12 streams, however cumulative distribution curves made with other minimum numbers of streams were similar. As previously discussed, the global variation among all streams is much higher then the popularity variation for streams of a same uploader. This result shows the importance to take into account each uploaders information to provide better popularity prediction for live content. However, predicting the precise number of viewers a stream will gather is beyond our reach, preventing us to be able to provision the exact amount of needed resources.



(a) YouTube



(b) justin.tv

Figure 4: Uploaders CDF, with at least 12 streams, of peak viewers coefficient of variation.

C. Number of servers versus bandwidth usage trade-off

In our model, a server can only serve a limited number of viewers depending on its bandwidth capacity and the bitrate of the served streams (see Section II). A naive approach consists in assigning new streams to a server until it reaches its maximum capacity. However, video streams popularity varies upon time. An already loaded server can thus acquire new viewers for streams it already broadcasts, it may then exceed its capacity. In order to keep offering an acceptable quality of service, it is necessary to use a secondary server, either a new one, or one already serving other streams (but with capacity left). The original server still has to broadcast the stream to the viewers it already served, but it also has to send it to the secondary server, using a preserved bandwidth dedicated to this use (in red in Figure 5). This stream "replication" mechanism is illustrated by Figure 5. Even if all the viewers can have an acceptable quality of service using this mechanism, it has a non negligible cost in terms of bandwidth. We can measure this extra cost by summing the quantity of data transferred across servers for stream replication, in bytes transfered.

As the number of viewers for a stream varies upon time (either increasing or decreasing), it is possible that the phenomenon described above leads to configurations for which many streams have to be transferred among servers. In this paper, we do not take into account the possibility to reconfigure

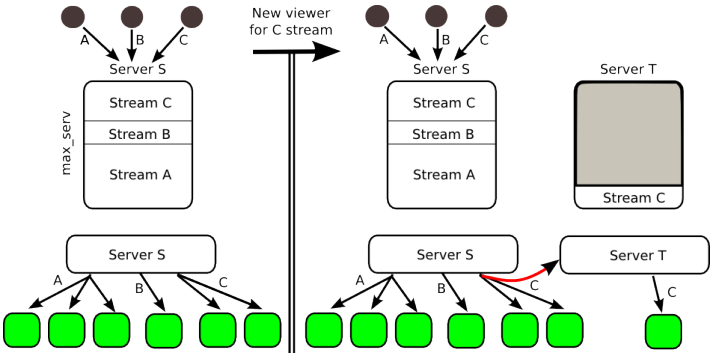


Figure 5: Server S has reached its maximal load, a new viewer for stream C implies a transfer on a new server (T).

the mapping: once a viewer is mapped on a server for a given stream, it will always receive this stream from this server. A reconfiguration would consist in migrating viewers among servers.

The simplest manner to limit the necessity to transfer/ replicate streams among servers is to over-provision, to provide a *margin* as illustrated by Figure 6. It allows the server to accept new viewers for the streams it already broadcasts, diminishing the risk to have to replicate a stream to a secondary server. Of course, the extra bandwidths provisioned for different streams on a same server are merged: at the start of a stream, extra bandwidth is provision, however, this extra bandwidth can be used to serve any stream broadcasted by the server. Indeed, the goal is to avoid a much as possible to replicate a stream on a secondary server. If the margin provisioned for a very popular stream is already used ant if it remains unused bandwidth capacity (provisioned at the start of other streams), it would be a bad idea not to use it for the very popular stream if it needs it (the other streams on the server may not need it).

However, provisioning extra bandwidth reduces the number of streams that can be served by one server, it thus implies to allocate more servers. The trade-off between the number of servers used and the inter-server bandwidth extra-cost can be tune through the margin value. Other parameters are also important, the server choice policy made by the dispatcher, *bestfit* or *worstfit*, can have an impact on the number of servers and on the inter-server bandwidth consumption.

D. Taking video streams popularity into account

The value of the provisioned margin has an important impact on resources consumption. If it is too low, many streams will have to be served by multiple servers, consuming a lot of inter-server bandwidth, as argued above. On the opposite, if it is too high, the system will involved many servers using only a small part of their capacity. Therefore it is important,for each stream, to set a margin close to what will effectively by needed.

Our approach, *POPS*, is based on the use of an estimator. It gives an estimation of the future popularity of a stream which is used to set the margin. The goal is to have a margin large

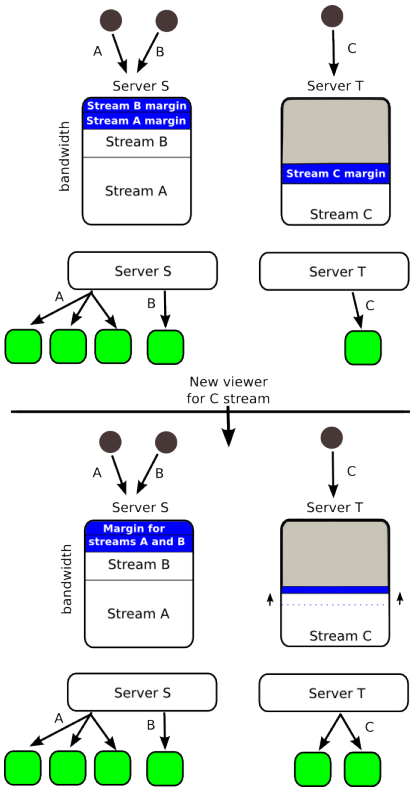


Figure 6: The server S has reached its maximal load (taking margins into account). It will not accept to serve a new stream, however, a new client for stream A or stream B can be served using part of the provisioned margins.

enough to diminish the risk to have to transfer the stream to another server during popularity peaks, but not too large to limit resource (server) waste. This is based on the finding that the popularity of the video streams broadcasted by a same uploader does not vary a lot. When a known uploader (having already at least one previous uploaded stream in the system's history) uploads a new video stream, our approach aims at provisioning enough resources to support its estimated viewer peak. We consider three different estimators:

- POPS estimates that a video stream will have the same popularity as the previous video stream uploaded by the same uploader.
- A-POPS (for adaptive POPS) has been designed to take into account uploaders having a growing popularity. If the history contains at least two video streams from one uploader, *A-POPS* expects the popularity of this uploader to continue to grow exponentially and computes the viewer peak of the next peak accordingly. If it can not estimate the growth (e.g., missing historical data), *A-POPS* adds an arbitrary 10% margin to the last popularity peak of the same uploader.
- Oracle is the estimator that could be build if it was possible look in the future. It knows in advance the number of viewers that a stream will gather, it can provision exactly the amount of resources that will be needed.

It is important to notice that even with a perfect oracle, this solution does not give the optimum: it could be possible to serve a short stream outside the peak load period using a portion of the provisioned margin. To do so, it would be necessary to be able to predict streams arrival times, streams lengths and the evolution of the number of viewers during the stream life. However, while predicting an approximate popularity peak for a stream is doable (because the popularity linked with the uploader of the stream), predicting when the peak of viewers will occur in the stream life and for how long remains beyond our reach.

IV. Evaluation

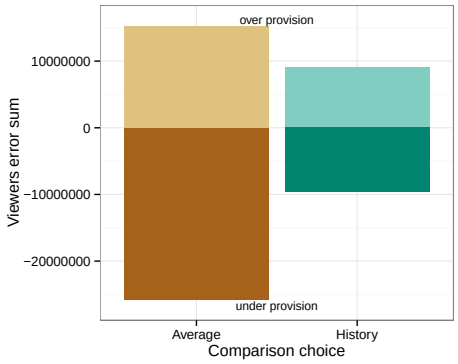
As discussed in Section III we have identified some trends on the uploader popularity. Our evaluation starts by showing how much taking the uploader popularity into account is important. To do so, we compared two alternatives for the uploaders popularity prediction: (i) a very naive approach using the global average number of viewers per stream (called *average*), and (ii) an approach that assumes that the next streams of the same uploader will have a similar popularity then the previous ones, based on A-POPS (called *history*).

Figure 7 presents the comparison between the two approaches. We called *viewer error* the difference between the prediction and the real values (the real number of viewers). For the *average* approach, the prediction values are the services averages which are 81 for YouTube and 21 for justin.tv. For the *history* the prediction for one stream is calculated based on the previous streams from the same uploader. We consider that the popularity for one stream will be the same as the previous one times the increase (if any) between the two previous streams (A-POPS). We present in the figure the sum of the viewer errors over the period. The positive values represents the over-provisioning of both predictions and the negative values the viewers not served. This figure confirms that a simplistic approach, such as using a global average, does not fit live popularity prediction and that an uploader history can indeed improve predictions and therefore the delivery provisioning.

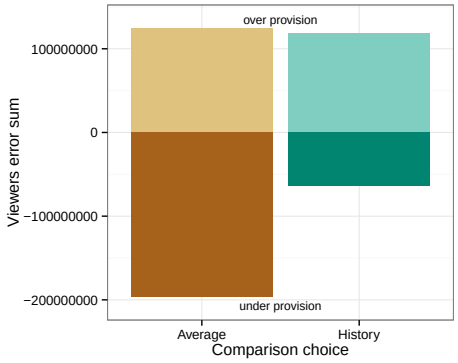
Parameters	Values
Traces	justin.tv, YouTube
MAX_SERV	$2Gbps$
Allocation Policy	FirstFit, BestFit, WorstFit
Approaches	Replication, POPS, A-POPS, Oracle
Stream Margin	0%,10%,...,50% of MAX_SERV

Table II: Simulation parameters

To further evaluate the approaches we set up several parameters summarized by Table II. We used the traces collected in the justin.tv and the YouTube platforms. The maximal broadcasting capacity MAX_SERV of each server has been set to $2Gbps$, which is a conservative value considering nowadays network cards. This limit is $2Gbps$ of outgoing bandwidth for each server where for each stream we consider the bitrate as $2Mbps$. We calculate the server cost in $servers-h$: one server used one hour counts for 1 $server - h$ and the network cost in bytes transfered between servers.



(a) YouTube



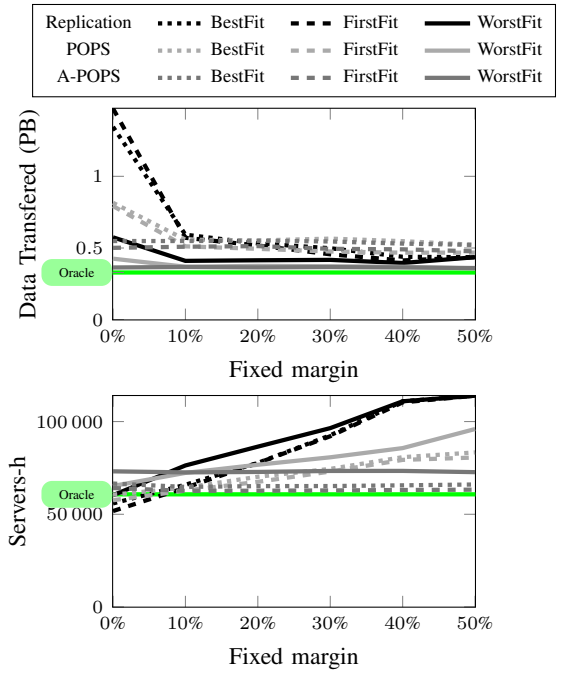
(b) justin.tv

Figure 7: Sum of the viewers provision errors, comparison between global average viewers and history

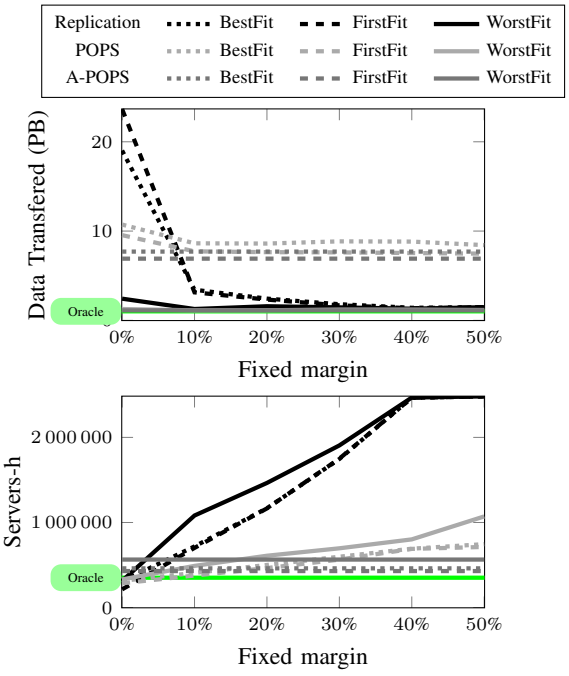
To choose among multiple servers to which one a stream will be given, we have evaluated the three well known allocation policies: FirstFit, BestFit and WorstFit. Concerning server provisioning, we have evaluated four approaches: *Replication*, *POPS*, *A-POPS*, and *Oracle*. *Replication* consists in the basic strategy using a *fixed* margin for each stream instead of a popularity estimator, illustrated by Figures 5 and 6: when the bandwidth needed for a stream (viewers * bitrate) exceeds the fix margin, the server forwards the stream to another server. *POPS* is our approach in which the margin is estimated considering that uploaders have a constant popularity. *A-POPS* takes into account uploaders popularity growth as described in Section III. Finally, *Oracle* gives the results *POPS* could have if the prediction was perfect (for each stream, we provision exactly the needed resources to face the viewer highest peak). This is possible because we know in advance the trace used to fit our simulator.

We vary the fixed margin from 0% to 50% of the total server capacity. Those fixed margins are used for *Replication*, *POPS*, and *A-POPS* approaches. In the case of *POPS* and *A-POPS* the fix margins are used when no prediction can be done, in absence of uploader history. As our simulation started with an empty history, a fixed margin is used for the first streams of each uploader. For the next streams *POPS* and *A-POPS* will estimate the margin. The traces we use are limited in time (two

weeks). More the historical data covers a long period, more *POPS* and *A-POPS* will be able to use an estimated margin.



(a) YouTube



(b) justin.tv

Figure 8: Comparing approaches.

We compare all the strategies in Figure 8. For each one, we compute the server cost ($servers - h$) and the inter-server bandwidth cost (bytes transfered). An horizontal line shows the *Oracle* strategy cost.

The first result of Figure 8 is that the allocation policy (FirstFit, BestFit or WorstFit) has a great impact. For the *Oracle* strategy, we only show the best of the three policies, the WorstFit one, for legibility reasons. The BestFit policy, as expected, has the lowest server cost. The FirstFit policy has a similar performance. However, these two strategies lead to many inter-server transfers.

Another observation is that increasing the fixed margin induces a proportional increase of the number of broadcasting servers used. However, in terms of bandwidth, above 25% of margin, the gain becomes negligible.

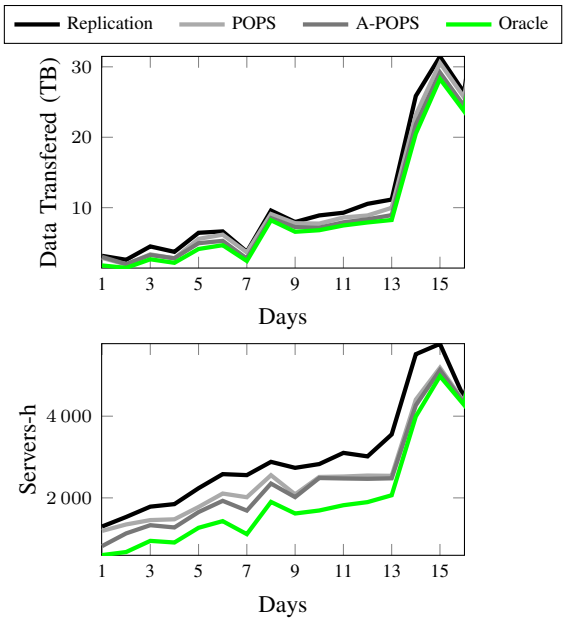
While comparing the results obtained with the different traces (justin.tv and YouTube) we see that that they are significantly different. This stresses the importance to work with multiple workloads. We can observe that the gain bring by *POPS* in terms of server cost is more important in the justin.tv case. This is due to the fact that justin.tv uploaders tend to produce more frequent streams while in the YouTube case the inter-broadcast delay of a given uploader is greater. In the case of justin.tv, the proportion of streams for which the system already knows the uploader is thus greater, allowing *POPS* to use more often its estimated margin instead of the fixed one.

However, we have detected that some justin.tv uploaders have an increasing popularity. Which means that their consecutive streams are more and more popular. *POPS* does not add an *extra* margin, thus, it is not tailored for such an increasing popularity. It just considers that a stream will have the same popularity as the last stream from the same uploader. This affects the bandwidth (bytes transfered) for justin.tv's FirstFit and BestFit policies. *A-POPS*, which takes into account the popularity growth, provides the ability to decrease the amount of bandwidth used (specially in the case of justin.tv) while having a cost close to the Oracle one in terms of servers, as illustrated by Figure 8.

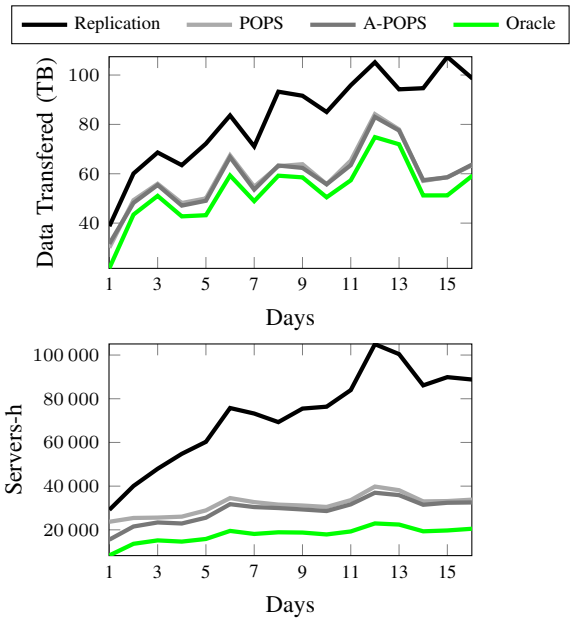
To better understand the first results, we studied the evolution of the system with the four approaches. Figure 9 presents the evolution during time. For each approach we selected their best set of parameters regarding bandwidth cost (WorstFit allocation with 20% margin). As expected, *POPS* and *A-POPS* efficiency improves along time, as the history becomes bigger, allowing them to use an estimation instead of a fixed margin like with the *Replication* strategy. In a real system, most *uploaders* are already known. Users broadcasting a video stream for the first time are rarely popular. However, our collected traces start at an arbitrary time in the system's life.

The benefits of *POPS* are significant and the results are similar to the ones obtain with the *Oracle* strategy on both traces. The *Replication* strategy performs poorly on justin.tv traces, where we have a large number of uploaders. The divergence between *POPS* and the *Replication* strategy, concerning number of servers cost, is due to the growing number of streams coming from already known uploaders; which is more important in the justin.tv case than in the YouTube one. *A-POPS* offers slightly better performances compared to *POPS*, thanks to the fact that it takes into account the uploaders popularity growth.

Many improvements are still to be done. For instance, it should be interesting to take into account an estimation of



(a) YouTube



(b) justin.tv

Figure 9: Approaches behavior upon time.

streams length. A study of growing popularity curves could also help the placement strategy by allocating small streams aside of longer streams that grows slowly.

V. Conclusion

The popularization of both networks and video capture devices has lead to the birth of user-generated content live streaming platforms. These platforms have to face a high number of simultaneous video streams that they have to collect and distribute to an even higher number of viewers. In this paper, we show that taking into account the future popularity of incoming streams is important while mapping them to a set of servers. This may help to prevent stream partition across multiple servers, inefficient in terms of inter-server network cost. We base our study on real datasets collected on the YouTube and the justin.tv platforms. We study the number of servers versus inter-server bandwidth usage trade-off. Our new approach, *POPS* uses popularity peak estimation of uploaders while placing new streams on broadcasting servers.

In this work we did not consider the possibility to migrate viewers across servers. At the end of viewer peaks, it should be interesting to migrate viewers and streams in order to be able to shutdown servers. This is part of our ongoing work. Furthermore, we plan to use enhanced mechanisms (*e.g.* learning) to try to predict the popularity evolution during the life of a stream.

References

[1] "Justin.tv," http://www.justin.tv/, (January, 2014).

[2] "Youtube," http://www.youtube.com/live/, (January, 2014).

[3] T. Hoff, "Gone fishin': Justin.tv's live video broadcasting architecture," High Scalability blog, Nov. 2012, http://is.gd/5ocNz2.

[4] A. Finamore, M. Mellia, M. M. Munafò, R. Torres, and S. G. Rao, "Youtube everywhere: impact of device and infrastructure synergies on user experience," in *IMC*. ACM, 2011.

[5] J. Zhou, Y. Li, V. K. Adhikari, and Z.-L. Zhang, "Counting youtube videos via random prefix sampling," in *IMC*. ACM, 2011.

[6] V. K. Adhikari, S. Jain, and Z.-L. Zhang, "Youtube traffic dynamics and its interplay with a tier-1 isp: an isp perspective," in *IMC*. ACM, 2010.

[7] X. Zhang and H. S. Hassanein, "A survey of peer-to-peer live video streaming schemes - an algorithmic perspective," *Computer Networks*, vol. 56, no. 15, pp. 3548–3579, 2012.

[8] A. Ganjam, S. G. Rao, K. Sripanidkulchai, J. Zhan, and H. Zhang, "On-demand waypoints for live p2p video broadcasting," *Peer-to-Peer Networking and Applications*, vol. 3, no. 4, pp. 277–293, 2010. [Online]. Available: http://dx.doi.org/10.1007/s12083-009-0059-1

[9] A. B. Vieira, P. Gomes, J. A. M. Nacif, R. Mantini, J. M. Almeida, and S. V. A. Campos, "Characterizing sopcast client behavior," *Computer Communications*, vol. 35, no. 8, pp. 1004–1016, 2012.

[10] H. Yin, X. Liu, T. Zhan, V. Sekar, F. Qiu, C. Lin, H. Zhang, and B. Li, "Design and deployment of a hybrid cdn-p2p system for live video streaming: experiences with livesky," in *ACM Multimedia*. ACM, 2009. [Online]. Available: http://doi.acm.org/10.1145/1631272.1631279

[11] M. Liebsch and F. Z. Yousaf, "Runtime relocation of cdn serving points - enabler for low costs mobile content delivery," in *2013 IEEE Wireless Communications and Networking Conference (WCNC)*, April 2013, pp. 1464–1469.

[12] H. Chang, S. Jamin, and W. Wang, "Live streaming performance of the zattoo network," in *IMC*. ACM, 2009.

[13] T. Qiu, Z. Ge, S. Lee, J. Wang, J. J. Xu, and Q. Zhao, "Modeling user activities in a large iptv system," in *IMC*. ACM, 2009.

[14] D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee, and K. K. Ramakrishnan, "Optimal content placement for a large-scale vod system," in *CoNEXT*. ACM, 2010. [Online]. Available: http://doi.acm.org/10.1145/1921168.1921174

[15] C. Jayasundara, A. Nirmalathas, E. Wong, and C. A. Chan, "Energy efficient content distribution for vod services," in *Optical Fiber Communication Conference*. Optical Society of America, 2011. [Online]. Available: http://www.opticsinfobase.org/abstract.cfm?URI=OFC-2011-OWR3