Konstantin Pussep, Osama Abboud, Florian Gerlach, Ralf Steinmetz, Thorsten Strufe:
Adaptive Server Allocation for Peer-assisted Video-on-Demand. In: Proceedings of the 24th IEEE International Symposium on Parallel and Distributed Processing IPDPS 2010, April 2010.

Adaptive Server Allocation for Peer-assisted Video-on-Demand

Konstantin Pussep*, Osama Abboud*, Florian Gerlach*, Ralf Steinmetz*, Thorsten Strufe§
*Multimedia Communications Lab, Technische Universität Darmstadt
Email: {pussep,abboud,steinmetz}@kom.tu-darmstadt.de
§Peer-to-Peer Networks, Technische Universität Darmstadt
Email: strufe@cs.tu-darmstadt.de

***Abstract*—Dedicated servers are an undesirable but inevitable resource in peer-assisted streaming systems. Their provision is necessary to guarantee a satisfying quality of experience to consumers, yet they cause significant, and largely avoidable cost for the provider, which can be minimized. We propose two adaptive server allocation schemes that estimate the capacity situation and service demand of the system to adaptively optimize allocated resources. Extensive simulations support the efficiency of our approach, which, without considering any prior knowledge, allows achieving a competitive performance compared to systems that are well dimensioned using global knowledge.**

I. Introduction

Peer-assisted Video-on-Demand (VoD) streaming systems are an attractive way to distribute video content through the Internet at low cost [1], [2]. They combine the scalability of Peer-to-Peer (P2P) systems, where users contribute their resources, and the service level guarantees of server-based systems.

A common solution is to organize peers in a mesh-based overlay where peers and servers contribute their upload capacity. For this, the video is split into segments that are initially downloaded from the content provider's servers, and later on redistributed by downloading peers.

A content provider utilizing a peer-assisted VoD solution aims at achieving the desired user streaming experience, while keeping the load at its servers as low as possible. Therefore, the provider should monitor the system behavior and adjust the contribution of the servers depending on the system performance.

However, such systems are highly dynamic because users arrive and depart from the system unexpectedly. Additionally, the resources of users are heterogeneous: some users have high bandwidth capacities and others quite low. This especially applies to the upload capacities, that are typically much lower than download capacities [3]. Thus, dynamic number of concurrent viewers and fluctuating capacities of peers should be alleviated by servers' contribution.

Unlike in pure server-based systems, server dimensioning in peer-assisted systems is more complex. The reason is that not only the demand but also the upload capacity of the system is dynamic. The main point is how much bandwidth can be contributed by peers and how much bandwidth must be provided by servers.

In this paper we address the issue of server allocation depending on system capacity deficit or surplus without knowing user behavior and capacities in advance. Additionally, the server bandwidth must be utilized efficiently to reduce the startup delay and stall times for video playback, which are the main quality metrics for the users.

To achieve this, we employ *adaptive server allocation policies* that allow available servers or peers with cached content to join and leave the overlay on demand. This way, servers can contribute their spare upload capacity otherwise, for example, to join other overlays with bandwidth demand, pro-actively upload some files, or even simply reduce costs if the upload is paid per volume. By minimizing the number of active servers the costs and overhead for the owners are reduced to the required minimum, while the users receive the desired quality of service as perceived in over-provisioned systems. We propose two policies, each one with a different metric to evaluate the streaming performance of the distribution overlay. Our performance evaluation analyzes the performance of the proposed adaptive policies, and further compares it with the static server allocation.

The paper is structured as follows: background on peer-assisted VoD is presented in Section II. Our system model is presented in Section III. In Section IV we present the adaptive server allocation approach and the details of the proposed policies. Performance evaluation is covered in Section V, while Section VI presents the related work. Finally, Section VII concludes the paper.

II. Background

We consider a typical scenario of a mesh-based peer-assisted VoD overlay where a content provider tries to reduce distribution costs by letting peers upload parts of the content. A media file is divided into segments and initially injected from content provider's servers.

The servers upload segments to a subset of peers which exchange these segments with each other and, therefore, contribute their upload capacity. Peers exchange the information about available and desired segments with each other and choose segments to download depending on their playback deadlines, that means, segments closer to the current playback position are prioritized. Typically, most urgent segments are managed in a high priority set whose size can range from few seconds to a minute of the video. Since the high priority set roughly corresponds to the playout buffer of the video player, peers are interested in keeping it full.

For the startup time we assume that peers start the playback once their playout buffer is filled and the remaining download time is lower than the duration of the video, to avoid playback stalling that occurs if segments close to the playback deadline are missing.

Peers are free to join and leave the overlay at any time and sometimes desired segments are not available in the neighborhood. Because of this, it is difficult to guarantee continuous segment supply for the whole download duration. *Prefetching* is a technique to smooth the streaming experience of users by downloading segments out of order. This way better segment distribution and bandwidth utilization are possible.

A special *indexing* server is utilized to track the active peers and to provide the peer lists for contact management. While the indexing server has the global view of the system, the information might be partially outdated due to the reporting intervals.

A. Give-to-Get Protocol

In order to utilize high capacity peers efficiently we utilize the Give-to-Get (G2G) system [4], a mesh-based protocol for VoD streaming. Here, uploaders prefer peers that turn out to be good forwarders. The video segments are divided into 3 sets: high, mid and low priority. The first one is the most crucial, since it corresponds to the playout buffer. Therefore, the main objective of the system is to fill this buffer fast and keep it filled during the playback, while segments from other sets are downloaded to enable prefetching.

Unlike the original G2G approach, we do not skip delayed segments of the video, but rather stall the playback until the playback buffer can be filled. This appears to be more suitable for a VoD application, since most users would prefer to see parts of the movie with delay instead of skipping them completely.

III. System Model

We assume that the content provider has access to a pool of servers, that can be either run by himself or rented, such as Amazon Elastic Compute Cloud (EC2)[1]. In such a scenario the content owner typically pays for the total volume of data uploaded from its servers, while Amazon also charges for usage time. Therefore, the provider is interested to avoid unnecessary uploads if the segments can be served by other peers. Uploading too few segments will result in unsatisfied users, while too much segments uploaded by the servers will result in unnecessary high costs. Furthermore, the reduction of unnecessary server online times allows content provider to reduce their energy consumption.

In order to estimate the bandwidth supply and demand in peer-assisted streaming we use the following notation:

- S: totally available servers (passive and active)
- S': active servers (subset of S)
- u_s: upload capacity per server
- L: peers active in distribution overlay
- u_l, d_l: upload and download capacity of peer l (in homogeneous case also denoted as u and d)
- r: video bitrate ($r \leq d_l$)
- $d_r = f * r$: required download speed ($r \leq d_r \leq d$), where f is the *prefetching factor* of the system
- g = peer upload utilization factor

As already explained in Section II prefetching is required to speedup the startup time and pre-load future segments. A suitable prefetching factor can differ depending on the utilized protocol. For example, in the G2G protocol f should be ≥ 1.2 [4].

On the other hand, the peer upload utilization factor g is the ratio of the average upload speed of peers to their upload capacity. Ideally, this values should be close to 1 but peers might be not able to exploit their upload capacity due to the lack of segments required by other peers (content bottleneck). In real systems utilization values around 0.8 have been reported [5].

We can easily compute the total required upload capacity as $R_{total} = |L| \cdot d_r$ and the total available upload capacity as $U_{total} = |L| * g * u + |S| * u_s$.

For acceptable streaming performance we then need $U_{total} \geq R_{total}$ and therefore

$$|S| \geq max(|L| \cdot \frac{d_r - g \cdot u}{u_s}, 0) \tag{1}$$

While the above model applies for homogeneous upload and download capacities, in the heterogeneous case, we obtain:

$$U_{total} = \sum_{l \in L} u_l \cdot g + \sum_{s \in S'} u_s \tag{2}$$

[1] http://aws.amazon.com/ec2/

$$R_{total} = \sum_{l \in L} r \cdot f = |L| \cdot f \cdot r \tag{3}$$

Our goal is to find the minimal subset $S' \subset S$ with:

$$\sum_{s' \in S'} u_{s'} \geq max(\sum_{l \in L}(d_r - u_l \cdot g), 0) \tag{4}$$

Here the right side expresses the missing upload capacity of the peers (under assumption that they cannot provide enough upload capacity) while the left side is the total contribution of the selected active servers.

We can immediately see from Equation 4 that in systems with the upload capacity of peers being very high (say $u_l \geq \frac{r \cdot f}{g}$) the server contribution becomes marginal. They are only needed to assure content availability and catch up with fluctuations in user demand and upload supply. However, even for the upload utilization factor $g = 0.8$ and low prefetching factor $f = 1.2$ we obtain that $u_l \geq 1.2 \cdot r/0.8$; so even for a moderate video playback rates of 512 kbps the average upload capacity of peers must be almost 800 kbps.

Ideally, the servers could contribute just enough upload capacity to balance out the available bandwidth in the system. Since this situation can change any time due to departure or arrivals of peers, the content provider must allocate server bandwidth to avoid serious degradations in user experience.

IV. Server Allocation Policies

In order to provide high Quality of Experience (QoE) for the users, our system utilizes the architecture shown in Figure 1. The indexing server monitors the peers' performance (step 1), determines the required upload capacity, and allocates servers (step 2) to join the overlay (step 3) and provide the missing upload capacity to peers in the most efficient manner (step 4).

Since users might abort the playback without watching the whole video, too aggressive prefetching might waste upload capacity. Therefore, exploiting the download capacity far beyond the video playback rate is not reasonable. Such peers can consume too much bandwidth and leave fast without contributing enough capacity in return. In order to avoid such unfair resource utilization, we limit the utilization of download capacity to a reasonable threshold, e.g, 2 or 4 times the video playback rate. On the other hand we limit the maximum upload utilization for high capacity users to the level of 2 times the playback rate.

A server allocation policy optimizes $S' \subset S$ according to the system parameters: the current number of peers $L(t)$, their upload and download bandwidth u_l and d_l, video bit rate r, and peer upload utilization factor g. These factors are measured by the indexing server, but might not be up-to-date due to the overlay dynamics. The factor f depends on the utilized streaming overlay and must be also taken into account while choosing S'.

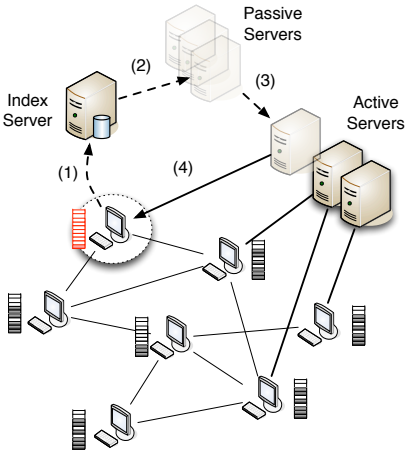


Figure 1. System Architecture.

An actual resource allocation policy is defined as $P = (M, \Delta, C)$ with the following components:

1) *Monitoring mechanism* M: this one collects the information about the overlay performance and state of single peers. This is done by the indexing server which is otherwise used for contact management. While the plain contact membership information is normally reported at a scale of several minutes, the streaming quality information must be updated more often to allow fast reactions on performance degradation.
2) *Decision metric* Δ: Different metrics can be used to decide when a server should join or leave the overlay. The candidate list includes the current server-to-peer ratio, missing upload bandwidth (difference between the total demand and the total available upload capacity), current download speed, and buffer states. We consider the (global) average download speed and playout buffer states as the most promising metrics.
3) *Connection management* C: Once a server decides to leave the overlay it can be done (1) immediately once the bandwidth excess was detected, (2) after finishing current transfers, or (3) once the connected peers are able to find new neighbors to replace the departing server. For the join procedure, the question is to which peers a new server should connect to. A possible improvement is also the decision to which peers to allocate the bandwidth.

Note that our system is managed by a provider, who controls the servers and the client software. In order to utilize the same architecture with open software and protocols, additional measures must be applied to prevent false reports from the users.

A. Global Speed Policy

The goal of this policy is to allocate minimal server resources necessary to balance the global overlay performance. To achieve this, the indexing server monitors the *global speed* G – the average download speed across all active peers. When the global speed falls below the desired level, additional servers are allocated to the overlay. On the other hand, if the measured speed is too high some servers are removed from the overlay.

Thereby, the target speed is defined accordingly to the desired prefetching factor f' as: $G_{target} = r \cdot f'$ Note that this value is independent from the actual number of the online peers $|L(t)|$. The actual demand for additional servers can be expressed as $(G_{target} - G_{measured}) * |L(t)|$ based on the recent peer reports.

In order to achieve good accuracy, peers must report their download performance frequently. We found one report in five seconds being sufficient for an appropriate estimation of the global speed. Then the indexing server computes the average global speed $G_{measured}$ over the last five seconds, compares it with the target speed G_{target}, and allocates servers according to their difference. The indexing server performs this computations in short periods and adds or removes only one server at once, to avoid too big performance oscillations.

A server that receives an instruction to leave, waits until the transmission of current segments is finished to avoid upload of incomplete segments. Contrary, a join action can be performed immediately.

The indexing server must process $|L(t)|/5$ client reports per second, that might become a bottleneck in case of thousands of files and ten thousands of users. Another possible drawback of this solution is that some peers might experience bad performance even if the global speed is balanced.

B. Supporter Policy

While the previous policy uses the average overlay performance as a metric to allocate server upload capacity, the supporter policy concentrates on the peers experiencing bad performance. Therefore, it addresses the possible drawbacks of the previous solution in order to avoid

- bad experience for a subset of users, even if the average performance is fine and
- too frequent status reports to the indexing server from too many peers.

Instead of periodic reports, a peer sends only the *starving* status message to the indexing server in the case when it cannot download high priority segments fast enough. Once the indexing server receives a certain amount of such requests in a given time, a passive server is selected to become a *supporter* (see Figure 1).

A supporter accepts only a reduced number of neighbors, so that it can provide segments at a speed greater or equal to the playback rate. In order to supply starving peers as fast as possible, a supporter exclusively serves the peers assigned to it. As long as the supporter has free upload slots, additional suffering peers can be assigned.

A naive approach could consider peers as suffering in the stall state only. Instead we try to identify suffering peers before the actual stalling happens. Therefore, we start *watching* peers if they miss segments in the playout buffer. A peer that enters the *watched* state sends a suffering report to the indexing server. Since missing segments in the playout buffer can have a transient nature, watched peers are considered as *starving* only if they were not able to fill their buffer in the time interval, (called *suffer time*) being an important parameter of the mechanism. In the starving state a peer will be connected to active supporters that currently serve less than *maxPeers*. If no free supporter can be found, a new supporter is allocated, but only if the number of unserved starving peers reaches the *minPeers* threshold. Note that this condition also covers peers being too long in the startup phase.

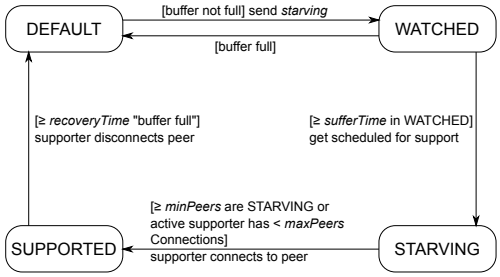


Figure 2. Downloader states in the supporter policy.

The algorithm to decide when and which peers should be considered as starving and receive help from supporters can be described as a peer state diagram with the following states (see Figure 2):

- *Default*: peer's playout buffer is full.
- *Watched*: peer is missing segments in the playout buffer recently.
- *Starving*: peer time in the watched state reaches the *sufferTime* threshold.

- *Supported*: total number of starving peers reaches the *minPeers* threshold.

Connected peers that left the suffering state and do not fall back in the *recover* time interval, are disconnected from the supporter. Finally, if all assigned peers have been served and no new peers have been assigned, the supporter leaves the overlay.

V. Performance Evaluation

In order to evaluate the performance of the proposed allocation policies, we implemented them in in an extended version of OctoSim simulator [6]. This discrete event-based simulator models data transfers at the level of file segments. As the underlying streaming protocol, we implemented in the simulator the G2G protocol (see Section II-A). All simulation runs were repeated ten times and provide average values together with their standard deviation unless mentioned otherwise.

A. Goal and Metrics

In order to show the feasibility and benefits of the proposed approach we conduct two groups of experiments: (1) performance and sensitivity analysis of policy parameters and (2) comparison of adaptive policies with static server allocation. We expected our policies to eliminate stall times and to perform at least as good as the static allocation policy with perfect predictions of user behavior. Furthermore, the supporter policy should be able to avoid too many outliers experiencing much worse performance than average users.

We use the following QoE metrics to evaluate the performance of the streaming overlay: *startup delay* until the video can be played and *stall times* during the playback. Thereby, we apply the startup solution proposed for G2G: the playback starts after the initial playout buffer (which corresponds to the high priority set) is filled and the remaining download time plus 20% overhead is smaller than the video duration. We consider especially the 50th and 95th percentiles of the delays, expressing the maximum delays experienced by 50% or 95% of users, respectively.

B. Basic setup

The basic scenario used for the performance evaluation is shown in Table I. It models a Video-on-Demand scenario where a content provider offers short clips, such as trailers or news reports, and tries to reduce its costs and increase system scalability by deploying a peer-assisted system. The content provider can decide in advance how many servers to allocate to a particular video or use our adaptive allocation policy. The parameterization of policies is described in the respective Subsections V-C and V-D.

Table I
Basic setup.

Parameter	Value
Simulation duration	30 minutes
Video length	5 minutes
Video bitrate	512 kbps
Available servers	up to 10
Server capacity (up)	2048 kbps
Peer capacity (up)	256, 512, 1024 kbps
Peer capacity (down)	2048 kbps
Peer distribution	0.3, 0.5, 0.2
Arrival rate (exp.)	6 peers/min
Playout buffer size	10 seconds
Departure rate	50% of video length on average

The peers are divided into three groups based on their upload capacities, representing slow (DSL), moderate (high-speed DSL or Cable), and fast (Ethernet) Internet connections similar to [7]. The relative size of these user groups is 0.3, 0.5, and 0.2, respectively. We also limit the maximum download capacity of peers to 2048 kbps to avoid fast peers consuming too much download bandwidth. In any case the download limit is four times the video playback rate and, therefore, sufficient even for an aggressive segment prefetching. Similarly, we limit the maximum upload capacity of Ethernet users to 1024 kbps to avoid too unfair resource utilization. Typically, even for high-end users greedy utilization of the upload capacity might result in throttling through the network operator. Each server has a upload capacity of 2048 kbps.

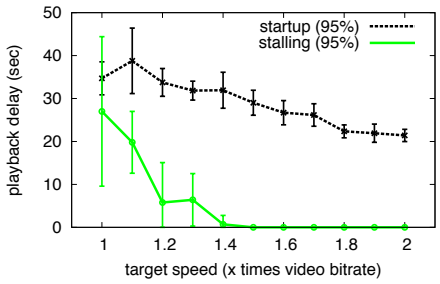
We model the session length of peers as follows:

$$T_{session} = min(T_{startup}+T_{playback}+T_{stall}, T_{departure})$$

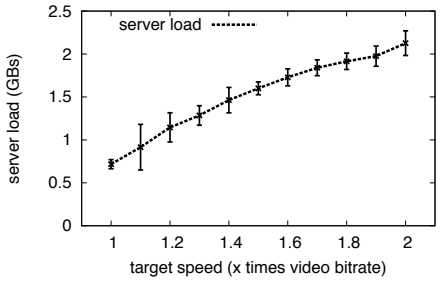
Here, users might stop the session without watching the complete video and, therefore, $T_{playback} \leq$ video length. The peers are enforced to upload the content as long as they watch it. We model the departure rate by letting peers on average abort the playback and depart from the overlay after 50% of video length after they start playing.

C. Global Speed Policy

In this subsection we analyze the performance of the speed-based adaptation policy with respect to the desired *target* speed. As already explained in Section IV-A this policy tries to keep the global speed close to the desired target where the target is defined as f' times video bitrate. If f' is set too low, some nodes might experience undesired playback delays. If f' is chosen too high, too many servers will join the overlay.



(a) Stall and startup delays (95th percentiles).



(b) Server load.

Figure 3. Impact of target speed factor on playback delay and upload.

We expect to find a suitable value in the range $[1 : 2]$, where 1 corresponds to no capacity for prefetching and 2 allows fast prefetching.

The basic parameterization of the policy lets clients report their performance to the indexing server once every five seconds. The indexing server then calculates the average speed over the five seconds interval.

In order to find suitable values of f' in the utilized VoD system, we run a series of experiments with varying parameter values. Fig. 3 shows the 95th percentiles of startup delay and stall times (Figure 3(a)) and the relative upload contribution of servers (Figure 3(b)) for different values of f'. We observe that f' values close to 1 results in higher delays and unstable performance (high standard deviation), while keeping the server's contribution low. Here peers whose average upload rate is close to the playback rate provide most of the resources but cannot keep up with the arrival and departure rates. On the other hand, high f' result in much lower playback delays, while the server load grows significantly. We further see that f' equal to 1.5 is sufficient to avoid stalling, and achieve low playback delay while avoiding unnecessary load at servers.

D. Supporter Policy

For the supporter policy we are interested in understanding the impact of the following parameters:

1) *minPeers*: How many suffering peers must be present to allocate an additional server,
2) *maxPeers*: Maximum number of peers a supporter can take care of,
3) *sufferTime*: Time interval with not full playout buffer to consider a peer as suffering,
4) *recoveryTime*: Time interval with filled buffer after which a previously suffering peer is considered as recovered.

To evaluate the impact of these parameters, we fix the default configuration as follows: *minPeers* = 1, *maxPeers* = 4, *sufferTime* = 5 seconds, and *recoveryTime* = 20 seconds. Then we subsequently modify single values.

Figure 4(a) shows the impact of the minimum number of suffering peers to allocate an additional server. We observe that the median is quite insensitive regarding this parameter, while the values around 2 and 3 peers prevent too bad performance for outliers. This can be explained by the fact, that waiting for too many suffering peers results in bad performance for single peers in the suffering state.

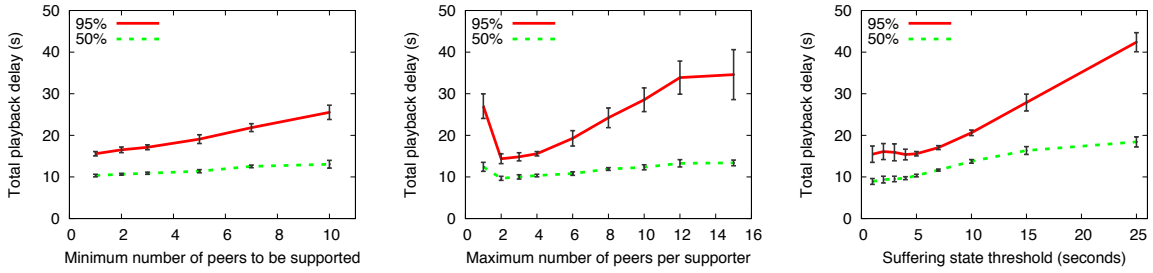
The impact of the maximum number of peers to be handled by one supporter is presented in Figure 4(b). We can observe that the best delays are achieved if the supporter handles only 2-4 peers. Beyond 6 peers per supporter the 95th percentiles increases dramatically. At the first spot the performance increase when going from 1 to 2 peers per supporter might appear counterintuitive. However, allocating the whole supporter capacity to a single peer results in too fast prefetching of segments and, therefore, wasted upload capacity, since a peer might depart without consuming and uploading them to other peers.

Another interesting point is how long a peer should try to fill its buffer being considered as *suffering* by the the indexing server. It turns out that values around 4 seconds are optimal (see Figure 4(c)). This is roughly the half of the playout buffer size and, therefore, corresponds to our expectation that supporter should react on the suffering state before playback stalls occur.

We also analyzed the impact of the recovery period with values between 1 and 20 seconds, but did not found a significant impact in our scenario. Both the server load and the startup delays were very close among the setup.

E. Comparison of Static and Adaptive Policies

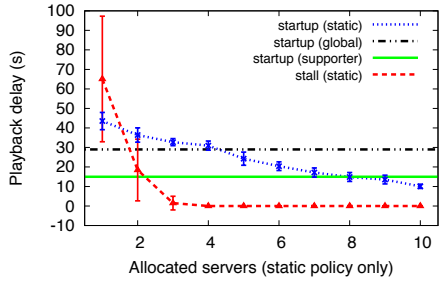
In this subsection we compare the performance of adaptive policies with the static server allocation in order to see if there are static setups that can outperform them. For the static policy we let 1 to 10 servers stay in the network for the whole duration of the experiment.



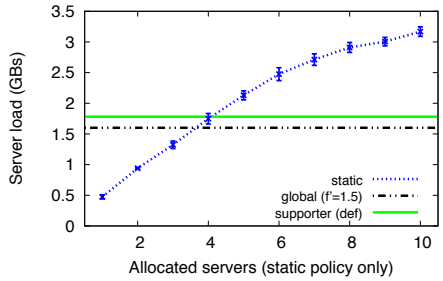
(a) minPeer threshold. (b) maxPeers per supporter. (c) Suffering time interval.

Figure 4. Impact of various parameters on the supporter policy performance.

The adaptive policies can allocate up to 10 servers on demand. For the supporter policy we use the default parameters as specified in Subsection V-D and for the global policy the target speed factor is set to 1.5.



(a) Stall and startup delays (95th percentiles).



(b) Data upload.

Figure 5. Comparison of static and adaptive policies.

The results for the playback delays are reported in Figure 5(a). The figure shows startup and stall times for the static policy, and only startup delays for adaptive policies, since no stalling took place (standard deviations for adaptive policies are the same as in V-C and V-D). We can observe that with the static allocation, the system performs best with 4 servers, though some peers still experience playback stalling. With too few servers the delays are high while with many servers the delays go down due to bandwidth overprovisioning. We also observe that the adaptive policies exhibit comparable performance, while the supporter policy outperforms the global speed policy (50% smaller startup delay). The probable reason is that supporters upload to suffering peers in the first place, while the global speed and static policies allocate the bandwidth to random peers. Table II additionally summarizes the average values for all three policies with the optimal configuration.

In Figure 5(b) we also show the data volume uploaded by servers. We can see that in the static configuration the server contribution grows almost linearly with the server count. With four static servers, they have to upload roughly the same amount of data compared with the adaptive policies. Though, the supporter policy uploads slightly more data than the global speed policy, it is able to achieve much lower startup delays (cf. Table II).

Table II
COMPARISON OF VARIOUS POLICIES (AVERAGE VALUES).

servers	server load	stalling (95%)	startup (95%)
static (4 servers)	1,75 GB	0s	31.0s
global ($f' = 1.5$)	1,60 GB	0s	29.0s
supporter (default)	1.78 GB	0s	15.5s

In summary, we observe that an adaptive policy allows a content provider to reduce costs by allocating available resources according to the overlay performance. This allows to achieve the performance of a well-dimensioned system without knowing the user demand and upload supply in advance. Even if the demand can be well estimated (here with 4 servers) the supporter policy deals better with the system dynamics and provides better streaming performance than the static and global policies.

VI. RELATED WORK

Recent work on how to deal with temporary undercapacity in P2P systems can be classified into two categories: (1) server allocation policies and (2) alternative proposals that do not allocate additional servers.

Regarding the allocation of servers in peer-assisted systems, different proposals have been done for file sharing (mostly BitTorrent) and live streaming systems [8], [9], [10], [11]. In case of BitTorrent, Das et al. [8] propose to estimate the server demand in order to guarantee minimal download speed to users. Similarly, Rimac et al. [9] consider the dimensioning of servers for single and multiple BitTorrent swarms. In AntFarm [10] a coordinator allocates peers and servers to swarms to provide minimal service level. However, such systems concentrate only on the download speed of peers, that is not constrained by the playback positions and buffer states. Therefore, they don't deal with the issue of startup delays and stall times. Wu, Li and Zhao present a prediction algorithm to estimate server bandwidth demand for peer-assisted live streaming [11]. Their *streaming quality* metric counts the number of peers that have a buffer count $\geq 80\%$. Differently to us, they don't address the delay and startup delays explicitly. Unlike in VoD, live streaming does not deal with the issue of prefetching and its interplay with the streaming quality.

Alternative approaches to avoid undercapacity start with advanced network coding, segment scheduling, and peer-matching algorithms to improve the throughput and capacity utilization for P2P streaming, e.g, [12]. Kumar et al. [13] state that the ratio of slow and fast users determine the P2P VoD performance and propose admission control and scalable video coding to deal with system's undercapacity. Garbacki et al. [14] propose the usage of helper peers in order to avoid bad user experience. However, they do not provide a metric how to calculate the desired number of helper peers. Huang et al. [1] analyze the impact of prefetching on peer-assisted VoD in surplus and deficit modes. They show that prefetching can significantly reduce the server load, especially when the bandwidth demand is close to the supply. Inspired by these results we propose to combine prefetching and adaptive server allocation to keep supply slightly higher than demand.

VII. Conclusion

We consider adaptive allocation of servers in peer-assisted Video-on-Demand streaming in order to avoid undercapacity and service degradation. To achieve this, we provide a simple demand model, and design two policies covering the demand monitoring, allocation decisions, and connection management for the servers. We show, by means of simulations, that adaptive policies can handle unknown user demand and provide high service level to the users while keeping the server load at a low level. On the other hand, focusing on suffering peers and supporting them preferentially can additionally improve service quality.

Acknowledgements

This work has been performed in the framework of the EU ICT Project SmoothIT (FP7- 2007-ICT-216259). The authors would like to thank all SmoothIT partners for useful discussions on the subject of the paper.

References

[1] C. Huang, J. Li, and K. W Ross, "Peer-Assisted VoD: Making Internet Video Distribution Cheap," in *International Workshop on Peer-to-Peer Systems (IPTPS)*, 2007.

[2] Yan Huang, Tom Z.J. Fu, Dah-Ming Chiu, John C.S. Lui, and Cheng Huang, "Challenges, design and analysis of a large-scale p2p-vod system," in *SIGCOMM*, 2008.

[3] M. Dischinger, A. Haeberlen, K. P. Gummadi, and S. Saroiu, "Characterizing Residential Broadband Networks," in *Internet Measurement Conference*, 2007.

[4] J.J.D. Mol, J. A. Pouwelse, M. Meulpolder, D.H.J. Epema, and H.J. Sips, "Give-to-Get: An Algorithm for P2P Video-on-Demand," in *MMCN*, 2008.

[5] B. Li, S. Xie, Y. Qu, G. Y. Keung, C. Lin, J. Liu, and X. Zhang, "Inside the New Coolstreaming: Principles, Measurements and Performance Implications," in *INFOCOM*, 2008.

[6] A. R. Bharambe, C. Herley, and V. N. Padmanabhan, "Analyzing and Improving a BitTorrent Networks Performance Mechanisms," in *INFOCOM*, 2006.

[7] C. Huang, J. Li, and K. W. Ross, "Can internet video-on-demand be profitable?," *ACM SIGCOMM Computer Communication Review*, 2007.

[8] S. Das, S. Tewari, and L. Kleinrock, "The case for servers in a peer-to-peer world," in *IEEE International Conference on Communications*, 2006.

[9] I. Rimac, A. Elwalid, and S. Borst, "On Server Dimensioning for Hybrid P2P Content Distribution Networks," in *Peer-to-Peer Computing*, 2008.

[10] R.S. Peterson and E.G. Sirer, "Antfarm: Efficient Content Distribution with Managed Swarms," *NSDI*, 2009.

[11] C. Wu, B. Li, and S. Zhao, "Multi-channel live p2p streaming: refocusing on servers," in *INFOCOM*, 2008.

[12] S. Annapureddy, S. Guha, C. Gkantsidis, D. Gunawardena, and P.R. Rodriguez, "Is high-quality VoD feasible using P2P swarming?," in *WWW*, 2007.

[13] R. Kumar, Y. Liu, and K.W. Ross, "Stochastic fluid theory for P2P streaming systems," *INFOCOM*, 2007.

[14] P. Garbacki, D.H.J. Epema, J. Pouwelse, and M. Van Steen, "Offloading servers with collaborative video on demand," in *IPTPS*, 2008.