Peer-Assisted VoD: Making Internet Video Distribution Cheap

Cheng Huang†, Jin Li†, and Keith W. Ross‡
†Microsoft Research, Redmond, WA 98052
‡Polytechnic University, Brooklyn, NY 11201

***Abstract*— We consider the design and potential benefits of peer-assisted video-on-demand, in which participating peers assist the server in delivering VoD content. The assistance is done in such a way that it provides the same user quality experience as pure client-server distribution. We focus on the single-video approach, whereby a peer only redistributes a video that it is currently watching. We first describe three natural prefetching policies for exploiting surplus peer upload capacity. We then study the performance of peer-assisted VoD using stochastic simulation and trace-driven simulation, with traces collected from the MSN Video service. The results of these simulations show that peer-assisted VoD, with the proper prefetching policy, can dramatically reduce server bandwidth costs.**

I. Introduction

Video-on-demand (VoD, also called on-demand video streaming) has become an extremely popular service in the Internet. For example, YouTube, a video-sharing service which streams its videos to users on-demand, has about 20 million views a day with a total viewing time of over 10,000 years to date [1]. Other major Internet VoD publishers include MSN Video, Google Video, Yahoo Video, CNN, and a plethora of copycat YouTube sites.

Most of the VoD being streamed over the Internet today is encoded in the 200-400 kbps range. At these rates, ISPs (or CDNs) typically charge video publishers 0.1 to 1.0 cent per video minute. It has been estimated that YouTube pays over 1 million dollars a month in bandwidth costs. These costs are expected to go up, as demand increases and higher-quality videos (with rates up to 3 Mbps or more) are made available.

In this paper, we consider the design and potential benefits of *peer-assisted VoD*. In peer-assisted VoD, there is still a server (or server farm) which stores all of the publisher's videos and guarantees that users can playback the video at the playback rate without any quality degradation. But in peer-assisted VoD, the peers that are viewing the publisher's videos also assist in redistributing the videos. Since peer-assisted VoD can move a significant fraction of the uploading from the server to the peers, it can potentially dramatically reduce the publisher's bandwidth costs.

There are two broad design approaches to peer-assisted VoD. In one approach, which we refer to as the *single-video approach*, a peer only redistributes the video it is currently watching; it does not redistribute other videos that it may have watched and stored in the past. The single-video approach is similar to a torrent in BitTorrent, in which all peers in the torrent share the same file. In the second approach, which we refer to as the *multiple-video approach*, a peer can redistribute videos that it has previously stored but is not actively viewing. Compared with the multiple-video approach, the single-video approach is simpler in the client and tracker design, and involves a straightforward end-user policy. In this paper, we focus on the single-video approach, which intuitively should provide good performance if, for each of the publisher's videos, many users are watching simultaneously. If the publisher has N videos and adopts the single-video approach, then the distribution problem essentially becomes N sub-distribution problems, one for each video. Henceforth in this paper, we focus on the design and analysis of peer-assisted VoD for single videos. The aggregation of N videos is shown through the multiplexing of all individual ones.

In a peer-assisted VoD system, when the peers alone cannot redistribute the video among themselves, the server makes up the difference, so that each peer still receives the video at the encoded rate. The server is only active when the peers alone cannot satisfy the demand. When the peers alone can satisfy the demand, not only is the server inactive, but the peers can potentially prefetch video from each other using the peers' surplus bandwidth. This prefetching allows the peers to build a reservoir of video, which can be tapped when the aggregate upload bandwidth of peers becomes less than the demand across all peers.

The contribution of this paper is as follows:

- We introduce the peer-assisted VoD problem and isolate the prefetching policy issue.
- For the single-video approach, we describe three natural prefetching policies for exploiting surplus peer upload capacity. These policies are *no-prefetching*, *water-leveling*, and *greedy*. We simulate each of the policies and show that they can significantly reduce server bandwidth usage, with prefetching providing dramatic gains.
- We then investigate the three polices using traces collected from the MSN Video service. We examine traces from the top two most popular videos. One has a high demand for a relatively short period of time; the other has a more moderate demand but for a relatively long period of time. We also examine the aggregation of the top 100 videos. The trace analysis again shows that peer-assisted VoD with prefetching can dramatically reduce server bandwidth costs.

II. Related Work

Most peer-assisted VoD solutions belong to the category of the single-video approach. Cui et al. [3] proposed oStream, which extended the application layer multicast to include buffer in the peer node to support VoD. Hamra et al. [2] proposed a tree based approach to implement peer-assisted VoD. In their scheme, all peers form a distribution tree, and based on bandwidth availability, a new coming peer is connected to a proper parent node in the distribution tree. Shen et al. [5] combined multiple description coding (MDC) of video with data partitioning, and provided a VoD solution with graceful quality degradation when peers fail and substreams of video are lost. Annapureddy et al. [6] studied using network coding in the VoD scenario. Li proposed PeerStreaming [4] to support the multiple-video approach. Using erasure resilient coding (ERC), PeerStreaming may partially cache previously viewed videos with the portion of the videos

being cached proportional to the upload bandwidth of the peer; which helps to reduce the overall cache requirement.

Peer-assisted video distribution is also catching up in the industry. According to a market report [8], in China alone, over 12 million Internet users have accessed peer-assisted streaming services or downloaded peer-assisted streaming softwares. At least 15 organizations are providing peer-assisted video delivery services. Though most services are peer-assisted file download or peer-assisted broadcast, peer-assisted VoD is certainly gaining popularity. Roxbeam [9], used to be called CoolStreaming and regarded as the first practical P2P streaming system, has recently added VoD in its service. UUSee [7], a new startup invested by SIG, provides thousands of VoD programs on its website. It has also signed a four-year contract with CCTV, the largest TV producer and broadcast network owner in China, to provide programs to all Internet users.

III. Surplus and Deficit Modes

To gain insight into peer-assisted VoD, we begin with a simple model. Let the length of the video (in seconds) be denoted by T and the rate of the video be denoted by r (in bps). Let the user arrive at the system in a Poisson process with parameter λ. Let M denote the number of user types, where the type m user is of upload bandwidth w_m and appears with probability p_m. Using the property of the compound Poisson process, the above user arrival model is the same as if each type m user arrives in a Poisson process with independent parameter $p_m\lambda$. Naturally, the average upload bandwidth of all users is $\mu = \sum p_m w_m$.

It follows from Little's law that in steady state the expected number of type m users in the system is given by $\rho_m = p_m \lambda T$. Thus, in steady state, the average demand is $D = r\sum \rho_m = r\lambda T$ and the average supply $S = \sum w_m \rho_m = \mu \lambda T$. We say that the system is in the *surplus mode* if $S > D$; and in the *deficit mode* if $S < D$. That is, the system is in the surplus mode if $\mu > r$, and in the deficit mode otherwise. It is important to note that even if a system is in surplus mode, at any given instant of time, the server may need to be active and supply video to peers. This is because (i) although on average the system is in the surplus mode, due to fluctuation, at a given instant of time the supply may become less than the demand; and (ii) it may not be possible to use all of the supply bandwidth at any given instant of time. The latter point will become more clear when considering the different prefetching policies subsequently.

IV. No-Prefetching Policy

We start with the simplest scenario in which all users adopt a *no-prefetching* policy. Under this policy, users only download content in real-time (the download rate equals r) and do *not* prefetch for future needs.

Suppose at a particularly instant of time there are n users in the system. Order these n users so that user n is the most recent to arrive, user $n-1$ is the next most recent, and so on. Thus user 1 has been in the system the longest. Let u_j, $j = 1, \ldots, n$, be the upload bandwidth of the j^{th} user and its probability be $p(u_j)$. Recall that user j is of type m with probability p_m, so $p(u_j = w_m) = p_m$. Let the state of the system be $(u_1, u_2, \cdots, u_n)$ and the rate required from the server be $s(u_1, u_2, \cdots, u_n)$. Since there is no prefetching, the demand of user 1 can only be satisfied by the server, which is the video rate r. Then, the demand of user 2 will be satisfied first by user 1 and then the server if u_1 is not sufficient. The demand of user 3 is satisfied first by user 1, user 2 and then the server, and so on. The following example might help clarify the description.

- For $n = 1$, we have $s(u_1) = r$.
- For $n = 2$, we have $s(u_1, u_2) = r + \max(0, r - u_1)$.

In general, for a given state, the rate required from the server is

$$s(u_1, u_2, \cdots, u_n) = \max_{1 \le j \le n} (r + \max(0, (j-1)r - \sum_{i=1}^{j-1} u_i)). \quad (1)$$

Note that in the above system, the upload bandwidth of the most recent user (user n) is *not* utilized. Furthermore, if $u_{n-1} > r$, the upload bandwidth portion $u_{n-1} - r$ of the next most recent user (user $n-1$) is also wasted, as it can only upload r to user n. Alternatively, if each user adopts a sharing window and can tolerate slight delay, then users arrival very close (e.g., users n, $n-1$, $\cdots$, $n-k$, for some k) can potentially upload different blocks in their windows to each other. Then, all users' upload bandwidths could be fully utilized. This is especially true in many other types of peer-to-peer applications, such as MutualCast [10], live streaming [11], etc. Nevertheless, the no-delay assumption serves as a good bound on how a simple and straightforward peer-assisted VoD system might perform. Furthermore, our analysis results show that it does *not* have much impact on performance, so long as the system scale is not too small.

For Poisson user arrival, it can be shown that the average additional server rate needed is given by

$$s = \sum_n \frac{(\lambda T)^n}{n!} e^{-\lambda T} \sum_{u_j} p(u_1, u_2, \ldots, u_n) s(u_1, u_2, \cdots, u_n) \quad (2)$$

where

$$p(u_1, u_2, \cdots, u_n) = \prod_{1 \le j \le n} p(u_j),$$

Although this result is not in closed form, s can be readily calculated using Monte Carlo simulation (details are omitted due to space constraint).

Now we provide results for the no-prefetching policy. We are interested in two aspects of a system: 1) the server rate with respect to the supply-demand ratio; and 2) the server rate with respect to the system scale. For simplicity, we assume that there are only two types of users in the system, where $w_1 = 768$ kbps, $w_2 = 256$ kbps, $r = 512$ kbps and $T = 300$s.

A. Surplus System

We tune the percentage of the type 1 and 2 users to drive the operation of the VoD system into surplus or deficit. Figure 1 shows the server rate in a surplus system. Each curve in the figure corresponds to a fixed supply-demand ratio and different points on a particular curve correspond to different number of concurrent users in the system.

We make the following observations. First, when the supply S is greater than the demand D by a substantial margin ($S/D = 1.4$ and above), the server rate is very close to the video bit rate r and does *not* increase as the system scales (i.e., the number of users grows). In other words, when there is sufficient average surplus in the system, an approach as simple as the no-prefetching policy can be adopted and the server rate will remain very low. Absurd as it sounds, this condition is totally realistic. Today's online video offerings typically use an encoding rate r that puts

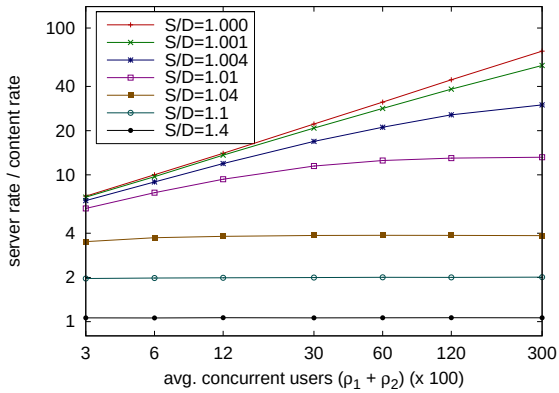


Fig. 1. Server rate in a surplus system

systems in the surplus mode (the argument will become more convincing after we present the real-world data in a later section). Second, even with little average surplus, the simple approach can greatly reduce the server rate. For instance, when $S/D = 1.04$ and the number of concurrent users is $30,000$, the server rate is about $3.8r$. Compared to traditional client-server models, where the server streams all data and thus its rate would be about $30,000r$, the bandwidth saving is significant. Third, the server rate increases significantly as $S/D \to 1$. The simple no-prefetching policy shows its weakness when the system operates closer to a balanced system.

B. Deficit System

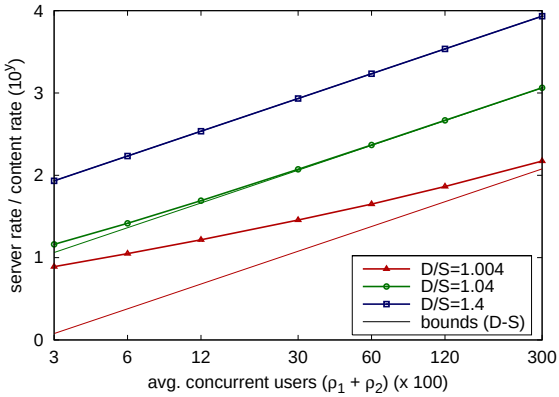


Fig. 2. Server rate in a deficit system

Figure 2 shows the server rate of a deficit system under various supply-demand ratios and number of concurrent users in the system. We have the following observations. First, when the supply S is less than the demand D by a substantial margin ($S/D < 1.4$), the server rate almost always equals to $D - S$. This means when the system is in this high-deficit mode, users do *not* need to adopt sophisticated prefetching approaches either. Note that the high deficit system is not the dominant operational mode for today's peer-assisted VoD yet. Second, the server rate deviates from $D - S$ as $S/D \to 1$. Third, the gap between the server rate and the bound $D - S$ shrinks as the number of users in the system increases. Nevertheless, if we consider the absolute value between the server rate and $D - S$, it is not negligible as the system scales up.

In summary, the no-prefetching policy provides insight into the surplus/deficit modes of peer-assisted VoD, and performs near-optimal in the high-surplus and high-deficit modes. It does *not* perform well in the balanced mode, where the average supply is approximately equal to the average demand. This motivates us to consider more advanced prefetching policies.

V. Water-Leveling Policy

The performance deviation from the bound in the balanced mode reveals a fundamental limitation of the no-prefetching policy. Due to the arrival/departure dynamics, at any given time, a balanced system (on average) might be instantaneously in a surplus or deficit state. When the system is in a surplus state, the no-prefetching policy does *not* use the surplus upload bandwidth that might be available. When the system enters a deficit state, the server needs to supplement the peers' uploading in order to satisfy the real-time demands. Intuitively, if users prefetch and *save for a rainy day*, the server contribution can potentially be reduced. In this section, we consider a *water-leveling policy*, for which users prefetch content and buffer data for their future needs. Of course, to keep the server rate low, we assume users do *not* prefetch from the server. A peer only prefetches from those peers that arrived before it and have sufficient upload bandwidth for distribution. Also, since users might have *savings* in their buffer, they can drain their buffer before they request any new data. Hence, the demand of each user might vary depending on its buffer level, as opposed to the constant demand with the no-prefetching policy.

In the water-leveling policy, we define $p_i(t)$, $d_i(t)$ and $b_i(t)$, respectively, as the playback point, the demand and the buffer point of user i at time t. The buffer point $b_i(t)$ is the total amount of content downloaded by user i up to time t. We always ensure that the buffer points of all peers follow its arrival order, i.e., $b_i \geq b_j$, for all $i < j$. We also define its *buffer level* as $B_i(t) = b_i(t) - p_i(t)$. Each user must maintains $B_i(t) \geq 0$ for continuous real-time playback. For simplicity, let the demand of user i be 0, if its buffer level is above 0, and r otherwise.

If all users maintain their buffer level above 0, the server rate would be 0, as the demands of all users are 0 at the moment. This observation leads to an insight that the server rate might be reduced if all users accumulate a high buffer level when the system is in surplus mode. Treating all users' buffer as *water tanks*, the *water-filling* strategy naturally suggests to fill the lowest buffer level first. Although this policy is sub-optimal, as users' arrival order also comes into play, it nevertheless, appears to be a very reasonable heuristic. The water-leveling policy can be implemented via the following three steps.

A. Satisfying Real-Time Demands

At some time t, assume there are n users in the system. The demand of each user is 0 or r, depends on whether its buffer is empty. As in the no-prefetching policy, we pass through all users in order and determine the required server rate. This ensures all real-time demands are satisfied. During the process, we record how much upload bandwidth remains at each user (denoted by l_i).

B. Allocating Growth Rates

After the first pass through all users, the water-leveling policy allocates remaining upload bandwidths to users with the smallest

buffer levels. It is easy to see that l_{n-1} (remaining bandwidth at user $n-1$) can only be allocated to user n, while l_1 can be allocated to any users from 2 to n. Due to this asymmetry, we perform the allocation of the remaining upload bandwidth in a backward manner, from user $n-1$ to user 1. There is no user later than user n, so its remaining upload bandwidth is not utilized.

Let g_i be the *growth rate* of user i, which represents the extra upload bandwidths assigned to user i beyond satisfying its real-time demand. We start with user $n-1$, and assign its remaining bandwidth l_{n-1} to user n, i.e., $g_n = l_{n-1}$. Then, we examine user $n-2$. The allocation of the remaining bandwidth at user $n-2$ can be calculated as (goes to user $n-1$ or n):

- $g_{n-1} = l_{n-2}$, if $B_{n-1} < B_n$;
- $g_n = g_n + l_{n-2}$, if $B_{n-1} > B_n$;
- $g_{n-1} = \min(\frac{l_{n-2}+g_n}{2}, l_{n-2})$, $g_n = g_n + l_{n-2} - g_{n-1}$, if $B_{n-1} = B_n$.

If $B_{n-1} \neq B_n$, then l_{n-2} is assigned to whoever has smaller buffer level. Otherwise, for $B_{n-1} = B_n$, the bandwidth assignment is to ensure that the growth rates of the user $n-1$ and n are equal after the allocation.

After the remaining bandwidth of user $n-2$ is completely assigned, we move to process user $n-3$ in a similar way. Note that the entire backward allocation can be completed with $O(n)$ time, as long as we maintain a simple auxiliary data structure to keep track of groups containing neighboring users with the same buffer level.

C. Adjusting Growth Rates

The growth rate allocation in the above step is purely based on the users' buffer levels. If the buffer point (note: not buffer level) of user $k+1$ catches up with user k (i.e., $b_{k+1}(t) = b_k(t)$), while user $k+1$ is assigned a higher growth rate than user k (i.e., $g_{k+1} > g_k$). Then, the buffer point of user $k+1$ will surpass user k. In such a case, we need to decrease the growth rate g_{k+1} to g_k. Hence, the third step is to pass through all users again in order, and shred off the growth rates of those users who have already caught up with earlier users, and re-assign extra bandwidths to later users. Again, as long as we update the auxiliary data structure (used in the second step as well) properly, this process can be completed with $O(n)$ time.

In short, the bandwidth allocation in the water-leveling policy consists three steps: 1) pass through all the users in order and determine the required server rate to support real-time playback; 2) process all the users backwards to assign remaining bandwidths; and 3) traverse all the users again in order and adjust the growth rates. The complexity of the entire allocation is $O(n)$.

VI. Greedy Policy

We carefully examined the simulation traces of the water-leveling policy. Typically when the server rate is positive, the buffer levels of earliest users (user 1, 2, 3, etc.) are usually 0. This implies that data demands imposed on the server are usually generated by the earliest users. Due to the asymmetry of the VoD system, only earlier users can upload to later users, and they are more likely to be assigned lower growth rates than later users. Hence, the actual behavior of the system is that later users tend to have higher buffer levels than earlier users, and earlier users have a higher risk of running out of buffer. Whenever that happens, these users will have to request data directly from the server. To correct this behavior, in this section we consider a *greedy policy*, where each user simply dedicates its remaining upload bandwidth to the next user right after itself.

The greedy policy works in the following two steps: The first step is similar to the no-prefetching policy and the first step of the water-leveling policy. We pass through all users in order and process each of them to determine the server rate that satisfy the real-time demands. We again record the remaining bandwidth at each user during the process. The second step of the greedy policy passes through all users in order and allocates as much bandwidth as possible to the next user. We still need to ensure that the growth rate of user $k+1$ does *not* exceed user k, when their buffer points become the same.

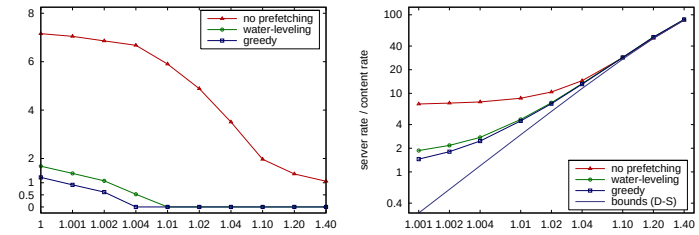
The second step can be further explained in the following pseudo code block. Note that the growth rate of the buffer point (demand + growth rate) is compared between user k and $k+1$. The allocation does *not* use the buffer level at all.

algo 1 2nd Step - Greedy Policy

```
budget := 0
for k = 1 : n − 1 do
    budget := budget + l_k;
    if b_k = b_{k+1} and d_k + g_k < d_{k+1} + budget then
        g_{k+1} := d_k + g_k − d_{k+1}
    else
        g_{k+1} := budget
    budget := budget − g_{k+1}
return
```

VII. Simulation Results

We developed a discrete-event simulator to study the performance of the water-leveling and greedy policies. Because no-prefetching is shown to be close to optimal in high-surplus and high-deficit modes, we focus on the cases that the average supply approximately equals the average demand (the balanced mode).



(a) Surplus mode (b) Deficit mode (x-axis logscale)

Fig. 3. Comparison of Three Prefetching Policies

We fix the user arrival rate at $\lambda = 1$ and vary the supply-demand ratio between $1/1.4$ to 1.4. We use a relatively low user arrival rate as such systems fluctuate more between instantaneous surplus and instantaneous deficit state in the balanced mode. From Figure 3, we see that the prefetching policies can result in dramatically lower average server rates. For example, in the perfectly balanced mode, prefetching can reduce the average server rate by a factor of five or more. In the surplus modes, the server rate actually goes to 0 when users are allowed to prefetch content. The buffer built up during the surplus state allows the systems to sustain streaming without using the server bandwidth

at all. In a deficit system, the server rate is much closer to the bound $D - S$. This is true in both the water-leveling policy and the greedy policy. Moreover, the greedy policy appears to achieve slightly lower server rates than the water-leveling policy under all the examined conditions.

VIII. Real-World Case Study

In this section, we use real-world traces to study peer-assisted VoD and the three prefetching polices. We use a set of trace records collected from the MSN Video service, which consists of all the on-demand streaming requests during the entire month of April 2006. Each trace record contains the start/end time (thus the duration) of a streaming session and the name/length/size of the accessed media file. All video streams were served by MSN servers (via a CDN, to be precise). Although these traces were generated from a client-server video streaming deployment, we can use them to drive a simulation and assess how well a peer-assisted VoD would have performed to save the server bandwidth.

A large number of videos offered by MSN Video are very popular, typically being viewed by many users simultaneously, which makes them good candidates for the single-video peer-assisted VoD distribution.

A. Inferring Users' Upload Bandwidths

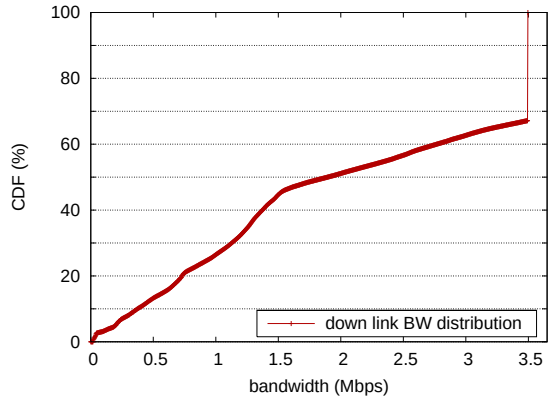


Fig. 4. User download link bandwidth distribution

Each trace record contains the download bandwidth of the corresponding user machine, which is measured by the Windows Media Server whenever a connection is established. Figure 4 shows the distribution of these measured download bandwidths. Note the streaming media server does *not* distinguish user download bandwidth greater than 3.5 Mbps, thus all such measurements collapse to a single point to the right of Figure 4. By using a simple mapping table based on available DSL/Cable offerings, we can infer the upload bandwidths of the users, as shown in Table I. Note that we are making very conservative

	modem	ISDN	DSL1	DSL2	Cable	Ethernet
downland	64	256	768	1500	3000	>3000
upload	64	256	128	384	768/384	768
share (%)	2.84	4.34	14.26	23.28	18.0	37.27

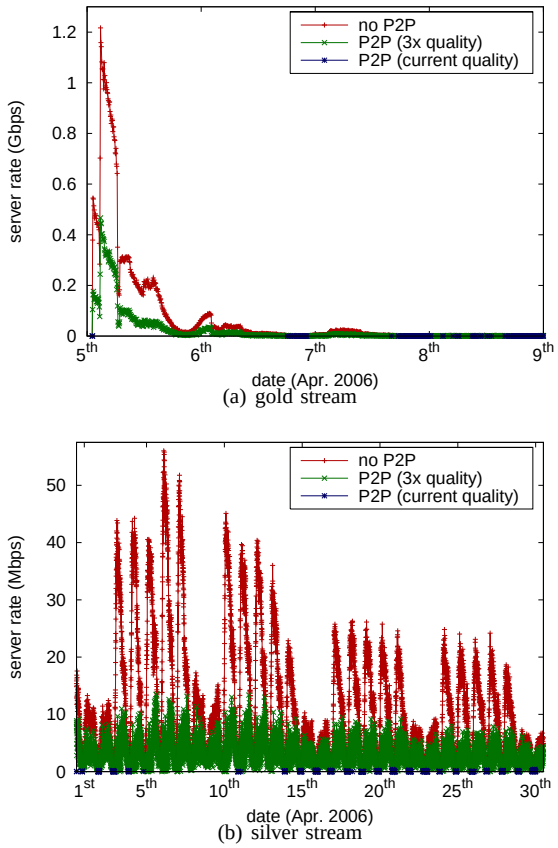
TABLE I
USER BANDWIDTH BREAKDOWN (KBPS)

assumptions on the users' upload bandwidth. For instance, many users, whose download bandwidths are above 3 Mbps, might actually be on university campus networks and therefore have much higher upload bandwidths than the 768 kbps assumed here. Nevertheless, a benign peer-assisted VoD system may not want to fully exploit the bandwidth of the high capacity users, as doing so might greatly deter their participation.

B. Trace Simulation Results

MSN served over $12,000$ on-demand videos that month. We first study the top two most popular videos. Each of the stream attracted about $800,000$ views during the month. We plot the server rate using the pure client-server deployment as the *no P2P* curve in Figure 5. Because the server rate is proportional to the number of users watching the video, this curve also reflects the popularity change of the video over time. The request patterns on these two streams are strikingly different. The most popular stream, which we call the *gold stream*, was released on April 5^{th} and quickly attracted a large number of requests. However, its popularity also declined very quickly, and after 5 days only occasional views were observed. The second most popular stream, the *silver stream*, remains quite popular throughout the entire month. We observe that its popularity goes through a peak and valley cycle each day. Interestingly, its popularity also has a 7-day cycle as well, where the valley matches nicely with weekend times.

We use these traces to drive the simulation and study the performance of the proposed policies. We first examine the pure client-server model and the peer-assisted model using the greedy policy. The performance of both gold and silver streams is shown in Figure 5. The server rate in the peer-assisted model is plotted



(a) gold stream

(b) silver stream

Fig. 5. Trace-driven simulation

as the *P2P (current quality)* curve. We also consider the case

when the video is distributed with peer assistance at much higher quality (its bitrate tripled), as shown in the *P2P (3x quality)* curve.

We make the following observations. First, it is clear that the MSN Video offering is currently operating in a surplus mode due to the relatively low bitrates of the video. We believe that this applies to other major VoD sites as well. Hence, if a peer-assisted VoD system were to replace the client-server system, the server rate would be dramatically reduced. In fact, Figure 5 shows a 1000-fold server rate reduction! Second, the occasional traffic often occurs at times when the video popularity enters valleys, which corresponds to small numbers of concurrent users in the system. This conforms quite well with our analysis. When the number of concurrent users is small, the peer-assisted VoD is more likely to run into a temporary deficit state and requires server participation. Third, when a peer-assisted VoD solution is deployed, we can easily offer much higher streaming quality (e.g. tripling the content bitrate) and still trim the server rate significantly. Finally, peer-assistance benefits both the flash crowd (gold stream) and the long-lasting (silver stream) scenarios.

We now compare the performance of all three prefetching policies using the MSN Video traffic, as shown in Table II. The system operates in the surplus mode with the current quality and in the deficit mode with 3 times quality. We report the

serv. rate (bps)	gold stream			silver stream		
	N.P.	W.L.	greedy	N.P.	W.L.	greedy
bitrate (kbps)	206			206		
no P2P (Mbps)	20.9			36.0		
cur. qual. (kbps)	225	173	172	258	11.9	0
3x qual. (Mbps)	7.03	7.00	7.00	7.60	7.25	7.19

TABLE II

TRACE-DRIVEN COMPARISON OF THREE POLICIES (USE THE 95 PERCENTILE RULE, A COMMON INDUSTRY MEASURE). N.P. - NO-PREFETCHING POLICY AND W.L. - WATER-LEVELING POLICY.

results using the 95^{th} percentile bandwidth rule, which works as follows. An average server bandwidth is measured every 5 minutes. All bandwidth points in the month form a set, and the 95^{th} percentile is the smallest number that is greater than 95% of the numbers in the set. This is a common industry measure used for billing purpose. We observe dramatic improvements going from client-server to peer-assistance with no-prefetching, and then further substantial improvements going from no-prefetching to either of the water-leveling or greedy prefetching policies. We further observe that the greedy policy tends to achieve the lowest server rate under all conditions. Note that the values of the gold stream are actually less than those of the silver stream, because it rarely generates traffic on days other than the 5 most popular days.

C. Multiplexing Effects with the Top 100 Videos

To get an estimation of the total server bandwidth costs, we have to take into account the multiplexing effect of many different videos, as one video's peak might happen to be another video's valley. To this end, we now examine the top 100 popular videos on the MSN site. We restrict ourselves to 100 videos simply to reduce the processing load. During the entire month of April 2006, these videos attracted over 16 million views in total. We calculate the aggregate server rate to support these 100 videos in three scenarios: not using P2P, using P2P without increasing quality and using P2P with 3 times quality. The average aggregate server rate is measured every 5 minutes. Then, the Cumulative Density Function (CDF) over the entire 30 days is plotted for each scenario. We then calculate the 95^{th} percentile bandwidth by making a cut at 95% on each curve.

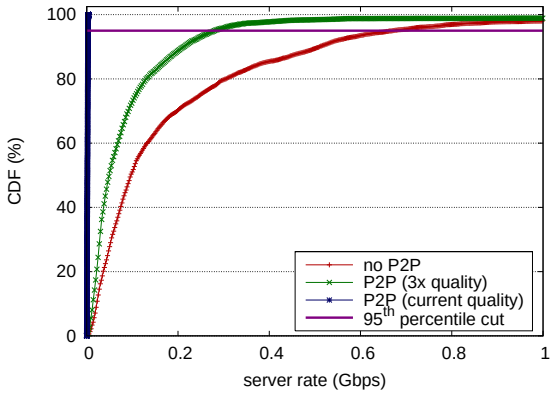


Fig. 6. System wide savings (aggregation of 100 videos)

As shown in Figure 6, the server rate without using P2P is 670.7 Mbps. Using peer-assisted VoD without increasing quality, the server rate becomes 2.82 Mbps. It is 280.3 Mbps using P2P with 3 times quality. Hence, by using peer assistance, the server bandwidth can potentially be trimmed down by 99.6% at current quality level. Alternatively, peer-assisted solution can triple the video rate (with a corresponding improvement in quality) and still trim the server bandwidth by 58.2%.

IX. CONCLUSION

We considered the design and potential benefits of peer-assisted video-on-demand. Using real-world data, we have shown that peer-assisted VoD can dramatically reduce the distribution cost of video publishers. There remains a number of open issues in peer-assisted VoD, including how close the proposed water-leveling and greedy prefetching policies are to optimal, and how much potential improvement can be gained using the multiple-video approach.

REFERENCES

[1] L. Gomes, "Will All of Us Get Our 15 Minutes On a YouTube Video?", *Wall Street Journal*, Aug. 30, 2006.
[2] A. Al Hamra, E. W. Biersack, and G. Urvoy-Keller, "A pull-based approach for a VoD service in P2P networks", *Proc. of HSNMC'04*, Toulouse, France, Jul. 2004.
[3] Y. Cui, B. Li, and K. Nahrstedt, "oStream: asynchronous streaming multicast in application-layer overlay networks", *IEEE JSAC*, vol. 22, no. 1, 2004.
[4] J. Li, "PeerStreaming: a practical receiver-driven peer-to-peer media streaming system", *MSR-TR-2004-101*, Sep. 2004.
[5] Y. Shen, Z. Liu, S. Panwar, K. W. Ross, and Y. Wang, "Efficient substream encoding for P2P video on demand", in submission.
[6] S. Annapureddy, C. Gkantsidis, P. R. Rodriguez, and L. Massoulie, "Providing video-on-demand using peer-to-peer networks", MSR-TR-2005-147, Oct. 2005.
[7] *UUSee*, http://www.UUSee.com
[8] *China P2P Streaming Research Report 2006*, iResearch, http://-www.iresearch.com.cn
[9] *Roxbeam*, http://www.roxbeam.com
[10] J. Li, P. A. Chou, and C. Zhang, "MutualCast: an efficient mechanism for one-to-many content distribution", *ACM SIGCOMM ASIA Workshop*, Apr. 2005.
[11] R. Kumar, Y. Liu, and K. W. Ross, "Stochastic Fluid Theory for P2P Streaming Systems", IEEE INFOCOM 2007, Anchorage, Alaska, May. 2007.