

CPM: Adaptive Video-on-Demand with Cooperative Peer Assists and Multicast

Vijay Gopalakrishnan*, Bobby Bhattacharjee†, K.K. Ramakrishnan*, Rittwik Jana*, Divesh Srivastava*
*AT&T Labs – Research †University of Maryland

Abstract—We present CPM, a unified approach that exploits server multicast, assisted by peer downloads, to provide efficient video-on-demand (VoD) in a service provider environment. We describe our architecture and show how CPM is designed to dynamically adapt to a wide range of situations including highly different peer-upload bandwidths, content popularity, user request arrival patterns (including flash-crowds), video library size, and subscriber population. We demonstrate the effectiveness of CPM using simulations (based on the an actual implementation codebase) across the range of situations described above and show that CPM does significantly better than traditional unicast, different forms of multicast, as well as peer-to-peer schemes. Along with synthetic parameters, we augment our experiments using data from a deployed VoD service to evaluate the performance of CPM.

I. INTRODUCTION

Video delivery is set to dominate Internet traffic in the near future and commercial entities have already announced or started Internet-based video services. These include network service providers (e.g., AT&T, Telefonica) and P2P-based service providers (e.g., SopCast, Peercast). We envision a future in which all content including television shows, user-generated videos, and full-length movies, will be viewed on-demand. Our work focuses on the scenario where IP is used for delivering the video-on-demand (VoD) content to homes and is complementary to streaming “live” content. We expect consumers to pay for this service, and in return expect video and service quality that is at least on par with current offerings on alternate non-IP delivery mechanisms, if not better. Towards that end, we introduce an architecture for provider-based delivery of VoD using Cooperative Peer Assists and Multicast (CPM).

Network service providers who own network infrastructure have the option to choose from or combine different data delivery mechanisms for delivering on-demand content. For example, streams could be just be unicast to each client from a server. Alternately, entire streams may also be multicast if the clients are willing to tolerate a (configurable) delay. This delay could be reduced using different methods such as unicast “patching”, through peer seeding, or by pre-populating the beginning of (popular) streams onto clients. Another viable alternative is also to use peer-to-peer approaches with entire streams cached on “seed” nodes, and then “swarmed” onto requesting clients using peer-transfer.

While there is an entire spectrum of choices, the mechanism adopted is usually dependent on the provider’s specific goals. From the provider’s perspective, cost is an overriding issue,

and hence infrastructure investment costs — in terms of server clusters and bandwidth — are important. From a client perspective, video startup latency and quality (delivering an uninterrupted stream) are most important. More importantly, from a service provider’s perspective, the mechanism should scale to large user populations, varying request arrival patterns including “flash crowds” of a large burst of users accessing one or a few popular pieces of content and accommodate a large content library. The choice of mechanisms is, however, made complicated because variables such as content popularity, request arrival, client population, system capacity, etc. that dictate the performance of these mechanisms change constantly. *Ideally, the service provider would need to deploy a single mechanism that automatically adapts to the prevailing conditions while enabling the providers to meet their goals.*

Our goal with CPM is to dynamically combine multicast, unicast, peer-transfer and pre-population in a seamless manner. The resultant system readily adapts to deployment constraints (ISP topology constraints, availability of multicast, peer-transfer capability, amount of available pre-population buffer), and to rapidly changing arrival distributions, content popularity, content length, and client populations. CPM is built around a *chunk transfer* mechanism — a simple RPC-like transfer for a “chunk” of data (about 30 seconds of video in our current implementation) — which enables it to rapidly adapt its delivery mechanism in response to changing conditions, without human or network-management intervention. CPM seeks to minimize required server resources (and network bandwidth), provide uninterrupted playback to clients with low startup delay, and not require large client-side storage or high-bandwidth client uplinks. CPM is cognizant of the network topology so that network resource usage is managed carefully. While the sub-components of CPM have been proposed elsewhere [1]–[6], the aim of our work is to harness the strengths of these basic mechanisms together to provide a single delivery mechanism that is adaptive to dynamically changing conditions.

We make the following contributions in this paper:

- We present the design of CPM, the CPM server, the server-side scheduling algorithm, and a description of how CPM adapts to different deployment constraints, user dynamics, and content distribution.
- We present simulation results across a variety of metrics, demonstrating the effectiveness of CPM compared to P2P schemes (modeled on the state of the art swarming protocols)

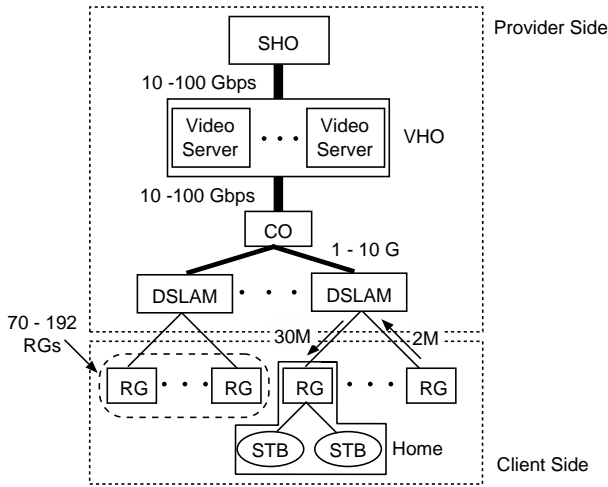


Fig. 1. IPTV network infrastructure and customer premise network

assisted by server unicast, and also to other “traditional” approaches such as unicast-only and multicast-only delivery. Using traces from a deployed VoD service, as well as synthetic ones based on the real world traces, we demonstrate the efficacy of CPM.

- We investigate a set of “what-if” analysis of future developments including scenarios such as higher definition video, much higher (symmetric) bandwidth to each home, and unbounded storage capacity at clients. We show that while existing schemes, especially those based on P2P approaches, work well for today’s work load, they will not be able to keep up when the prevailing conditions change.

The rest of this paper is structured as follows: We briefly describe a typical target infrastructure in Section II and present a detailed description of the CPM protocol in Section III. Simulation results are presented in Section IV. We discuss related work in Section V before concluding in Section VI.

II. TARGET NETWORK INFRASTRUCTURE

We describe a typical service provider network infrastructure layout that can be used to deploy CPM. Figure 1 shows the logical end to end architecture. These networks are typically hierarchical with the Super Head End Office (SHO) at the top of the hierarchy. The SHO is the primary point for content acquisition and distribution. Content acquired at the SHO is carried over a backbone to metro-region Video-Hub-Offices (VHO). The VHO houses several VoD servers that store and distribute content to the end users.

Each VHO is then connected to the access nodes through Central Offices (CO). These access nodes typically are connected by fiber-to-the-premise (FTTP) or fiber-to-the-node (FTTN). FTTP access nodes deliver services over an optical network directly to customers’ homes. Alternatively, in a FTTN deployment, Gigabit Ethernet fiber connections are used to connect the COs with DSLAMs (Digital Subscriber Loop Access Multiplexer) with each home connected directly to DSLAMs by copper lines. It is not uncommon for a CO to have several tens of DSLAMs subtending from it with each DSLAM

supporting up to 200 homes. Note that cable networks are quite similar. A typical metropolitan area has a few hundred thousand homes that are “passed” (only a subset of these are “served” as customers of the cable provider). There may be a hierarchy below each CO, with secondary hubs that may support a few thousand homes.

The client end comprises of a residential gateway (RG) and one or more set-top boxes. The RG provides a managed interface between the customer premises equipment and services delivered over the network infrastructure including VOIP service and broadband data access. Video streams are processed by the set-top boxes that are connected directly to the RG. While the set-top boxes will be quite heterogeneous in terms of their capabilities (including “dumb” ones), between the RGs and the set-top boxes (STB), we can safely assume that there is some available disk space (e.g., in the order of 1-100 Gbytes) and processing capacity. We anticipate that part of this storage and processing may be available for use by the service provider.

All the equipment in the provider’s side of the environment and the client RGs are IP Multicast capable and support PIM-SSM [7]. The STBs support IGMP and act as end-points for IP Multicast. Currently, Providers stream “linear” (live broadcast) TV using IP Multicast over this environment.

III. THE CPM SCHEME

The CPM protocol makes use of a combination of server multicast and peer-assisted downloads to serve videos on-demand. The architecture of CPM contains three major components: (1) a data model for video, (2) a protocol for content discovery and retrieval, and (3) a smart server-side scheduler. In this section, we describe the components of CPM in detail.

A. Data Model

In CPM, we divide a video into fixed sized *chunks*. Chunks are the smallest granularity of data transferred in CPM. Each chunk is identified using a globally unique identifier (GUID), and may have more than one GUID. A *segment* of video consists of a sequence of chunks. Each segment is also assigned a GUID which maps to a unique sequence of chunks. Finally, a *video*, consists of a sequence of segments.

Chunking, as an idea, is not new and has been used in a number of recent protocols [3], [8], [9]. Similar to these protocols, in CPM, chunks are points where a new download decision can be made. In CPM, a peer can switch from peer transfer to a server transfer for each new chunk. Smaller chunk sizes enable faster adaptation, but increase control overhead since each chunk needs to be individually located and accounted for. We have experimented with different chunk sizes and by default use 30 second chunks. This choice of chunk size is based on amortizing overheads while allowing for a reasonably small unit of content to be served up by any one entity in the system (thus enhancing the failure tolerance of the system).

Our complete data model supports transcoding (to support a variety of devices without storing content at different resolutions) and content search (for locating video using different

types of meta-data). However, only the chunking aspect is relevant for this paper and we do not elaborate on content search and transcoding.

B. CPM Data Transfer

In CPM, a complete copy of every video is present at each VHO (referred to as the server from now on). Clients may be prepopulated with a set of chunks, and cache a subset of chunks that they have played. Users interact with the client software and initiate playout.

Upon receiving a “play” request for a video, a search mechanism maps the user’s request to a video, which is then resolved to a sequence of segments, and in turn to a sequence of chunks that must be located and played. These sequences of chunks form the input to the CPM data transfer protocol. The client uses the protocol described below to locate and play each chunk in sequence. However, in practice, the client runs this procedure in parallel to obtain a small number of chunks simultaneously. Every chunk is transferred relatively independent of other chunks and the procedure described next is what is followed for each chunk transfer.

1) *Locating a Chunk*: Figure 2 shows a high-level overview of the set of steps a client C undertakes in locating a chunk k . C first checks its local cache. A chunk may be pre-cached because the segment (or video) was played earlier, the chunk was part of a different video (e.g. a mashup) that was locally played, or C was prepopulated with this chunk (refer to Section III-C).

If k is not available locally, C must fetch the chunk from the server or other peers. C first computes the chunk’s current *slack*, which is the difference between the chunk k ’s playout time and the current time. The slack is the amount of time C has to fetch this chunk without interrupting playout. Depending on the slack available, C may choose to go directly to the server or try a peer transfer.

As a first step, the client queries the server about an outstanding multicast request for this chunk. If such a request is already queued at the server, the client is added to the multicast group (and the group’s schedule adjusted appropriately as described in Section III-B2).

If there is no multicast scheduled, C uses the peer search protocol (Section III-B4) to find a set of peers \mathcal{P} that have the chunk stored locally. C contacts these peers in turn and tries to initiate a transfer as long as the peer’s available upload bandwidth is sufficient to transfer chunk k within its slack.

If \mathcal{P} was empty, or if none of the peers have sufficient upload bandwidth, the client contacts the server for k with the slack (the slack is continuously adjusted). The server schedules a new transfer for k and returns a multicast group ID that C joins. Once again, the server ensures that chunk k is transferred completely before the slack runs out.

When a chunk is not found locally, CPM clients strictly prefer server *multicast* over peer transfers, which are preferred over server unicast with the following rationale: since the server is already scheduled to transfer a chunk, the marginal cost for being added to the multicast is small (overhead of

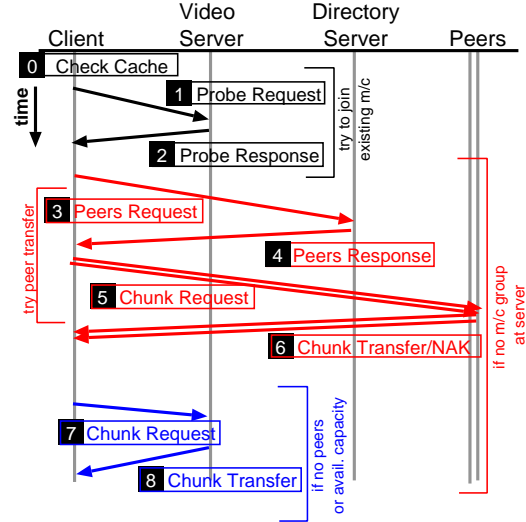


Fig. 2. Chunk transfer protocol schematic

group setup and forwarding at routers). While the peer transfer can be considered “free”, i.e., the peer transfer completely removes the server from the transfer path, we prefer server multicast in CPM since this leaves the peer open for serving another chunk (that is not being multicast) to a different client.

2) *Server-side Scheduling*: Clients contact the server \mathcal{S} for a chunk k when other peers don’t have it or cannot transfer k before its slack expires. In CPM, the server, by default, transfers chunks via multicast. For a given request from client C for chunk k , \mathcal{S} forms the multicast group as follows:

- \mathcal{S} computes a “deadline” for the current request using the slack information. The deadline is the maximum time by which transmission for the current request must begin (or else the slack will be violated).
- Each chunk request is mapped to a multicast group and each group has an associated “start time”. The start time of a group is computed as the minimum of the deadlines for all requests in that group.
- If there are no existing groups for this chunk, the server simply starts a new multicast group, sets the start time to be the request deadline, and informs the client of the group.
- If \mathcal{S} finds an existing group for chunk k and the current request’s deadline is greater than the start time of the group, then server simply adds the client to the multicast group and informs the client of the group.
- If the request has a stricter deadline than the start time of the group, then the entire group is moved “up”, i.e., the start time of the multicast group is set to the current requests’s deadline. Once again, the client is informed of the multicast group.

The algorithm described above assumes that the server is able to allocate a group whenever necessary. However, in practice, the server will have a fixed number of groups (due to the bandwidth restrictions at the server). While the server may not delay transmissions beyond the start time of a group, it may move them arbitrarily early in time (though doing so may cause the server to miss out on aggregating requests

it might have been able to fit into the single multicast). A server with bandwidth constraints will not be able to serve all requests with deadlines — this problem is equivalent to the broadcast scheduling with deadlines and profits problem, for which computing an optimal schedule is NP-Hard [10]. In our implementation, where the server is bandwidth constrained, we use a greedy heuristic to pack transmissions into the first available “gap”. If such a gap cannot be found, then the request is dropped. Finally, if the “multicast” group contains a single client, the server simply unicasts the request.

3) *Peers serving chunks*: Peers may serve any chunk they have cached. This includes chunks they were prepopulated with and chunks that were downloaded for playback. Peers receive a request with a chunk GUID and a slack. In CPM, we assume that the service provider provisions a fixed amount of peer upload bandwidth (and separate from the uplink available for data and voice traffic) for chunk transfers. If a requested chunk k is found locally, then peer p uses the slack value in the request and its available upload bandwidth to determine whether it can successfully serve k before k ’s slack expires. If it can, then p initiates a transfer for k ; if k is not found or the slack is not sufficient, the p returns an error.

4) *Chunk Search*: We have implemented two strategies — one distributed and one centralized — that clients use to find out which other clients have a chunk k .

In our distributed protocol, each client periodically multicasts a list of the set of chunks it has to nodes within its DSLAM. These multicasts enable each node to learn the set of chunks that the other nodes under the same DSLAM have. Within a DSLAM, the client with the lowest ID computes a digest (a Bloom filter) of the set of chunks cached at all clients in the DSLAM and periodically multicasts it to all clients under its CO. When a client wants to locate a chunk, it first tries to find a peer within its own DSLAM which has advertised the block. If unsuccessful, it tries to match the chunk ID against one of the Bloom Filters, and then contacts the node (say l) which advertised the Bloom Filter if there is a hit. Node l then forwards the request to one of the nodes within *its* (l ’s) DSLAM which has the requested chunk cached (assuming that the hit was not a false positive).

Our centralized scheme, which may be preferable to service providers due to policy reasons, uses a *directory server* (DS) co-located at the CO. The DS’s identity is provisioned into each client, and a client informs the DS whenever it receives a new chunk or deletes a chunk from its local cache. When a client searches for a chunk, it simply asks the DS, and the DS returns a small set (configurable number) of other clients, chosen uniformly at random, that have this chunk cached. All the results we present in this paper use the DS.

The DS can be implemented using a single host or a cluster as necessary. With 30 second chunks, a client searches for a chunk, on average, roughly every 30 seconds. With 10,000 active clients, arriving over one minute, the directory server will receive around 20,000 queries per minute (334 per second), which is reasonable even for a single well-provisioned host. The DS is a single point of failure. However, it is

amenable to simple replication, and traditional techniques can be used to make it a robust available service.

C. Optimizations

Prepopulation Prepopulation allows for videos to be started without delay and also increases the slack for subsequent segments. If the client has to go to the server, the slack allows requests to be aggregated, thereby increasing transfer efficiency. Prepopulation can occur during off-periods (e.g., during the day, or overnight) and does not require much storage on the client. In CPM, we prepopulate both the initial segments of movies and popular “jump-points”, which enables quick seek into popular parts of videos.

Server Bias When a client receives a chunk from the server through multicast, it implies that there are other clients also receiving that video from the server. Hence if the client “remained” in the multicast group, then the subsequent chunks would also be multicast from the server. Based on this intuition, the clients use the following rule: If a chunk is transferred from the server in a multicast group, then for the subsequent chunk, go back to the server directly. If and when it receives a chunk through unicast, the client goes back to the full peer search protocol.

IV. EXPERIMENTAL RESULTS

We have evaluated CPM’s performance and compared it against other schemes using a comprehensive set simulations. The first part of this section contains results that use request traces from a real-world VoD deployment. The latter set of results uses a synthetic data set to focus on the performance of CPM and other protocols when system parameters change.

A. CPM Implementation

We first implemented a proof-of-concept CPM client and video server. Our implementation contains both the distributed search protocol and a directory server. The server transfers chunks using IP multicast and implements the full server-side scheduling algorithm. We have demonstrated complete end-to-end functionality by transferring chunks from the server and peers and by playing out the video using standalone video playout software (`mplayer` in Linux).

Our implementation shows the viability of CPM. We, however, did not have access to a large enough testbed to fully investigate all aspects of CPM, or to compare it to other protocols. Instead, we replaced the implementations network I/O calls with stubs that interfaced with a packet-level, discrete-event simulator. In this section, we present results from this simulator.

B. Comparison to other protocols

Unicast We have implemented a baseline unicast transfer. The unicast simulations do not use peer transfer.

Multicast We have also implemented a multicast transfer protocol. The multicast server batches requests with compatible slack, but multicast clients do not use peer transfer. We

compare CPM with this form of multicast rather than with cyclic multicast since this protocol produced the lowest server overhead, regardless of the period of the cyclic multicast.¹

Server Assisted P2P IPTv using P2P is an active area of research. Instead of focusing on any single protocol, we have implemented a swarming protocol that incorporates ideas from a number of state-of-the-art P2P protocols.

Like all swarming protocols, our P2P protocol transfers videos in chunks. We assume that each client identifies which other peers have the chunks and downloads the chunks from these peers (possibly striping the chunk across peers). If the available client bandwidth is not sufficient or there are no peers having the chunk, then the chunk is unicast from a “server”. Our results quantify the amount of “server” bandwidth needed to satisfy client P2P requests, and is meant to represent the bandwidth required out of any combination of servers (either a single host or a distributed cluster) and seeders.

C. Metrics

In the results that follow, we evaluate the performance of each system across a number of metrics as we vary the parameters that affect performance.

- **Server Capacity:** Our primary metric is the maximum (I/O and network) bandwidth required at the server for each client to play out every request without interruption. For P2P systems, this is the measure of seed bandwidth; for all other systems, this is the amount of bandwidth the VHO must be provisioned with. We do *not* consider metrics such as “temporal continuity” which is a measure of the number and span of missing packets/blocks as a client plays out a video. Instead, for our intended deployment, we focus on the system requirements for bit-perfect delivery.
- **User Experience:** Since all videos are played in full, the primary user metric is startup delay, which we tracked for each scheme.
- **Protocol and network overhead:** CPM incurs protocol overhead due to searches and transfer requests, which we tracked. We did not include these last two due to lack of space.

D. Behavior with user requests of a commercial VoD service

We compare the performance of CPM with that of unicast, and P2P with one week worth of traces from a deployed VoD system of about 100K active subscribers making approximately 369K requests (timestamped at 1 second granularity) for 2600 requested videos (or 36 requests per minute). The top 60% of the requests asked for 9% of the videos with the next 30% of the requests asking for the next 25% of the videos. One can observe that the popularity of videos is quite spread out and is unlike what has been observed in a YouTube-like setting [11] where 80% of the requests were to 10% of the content. We attribute this to the relatively small size of the video library and to the new-ness of the service.

¹The peak server bandwidth for cyclic multicast depends on inter-arrival time of requests and cyclic multicast interval. When the cycle interval is long, the number of unicast transfers for patching increases. When the intervals are short, the number of parallel streams increases.

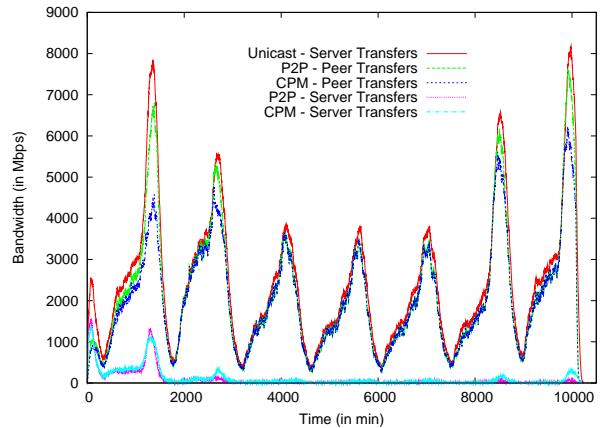


Fig. 3. Bandwidth required by different schemes to satisfy user requests following the real-world trace

We use this trace data to evaluate the bandwidth needed, over time, at senders (peers and server) in order for all clients to play their videos without interruption. We plot the results of this experiment in Figure 3. One can immediately observe that, as expected, unicast depends on the number of requests and the bandwidth needed at the server follows the daily request arrival trends, with a peak server requirement of about 8 Gbps. In contrast, both CPM and P2P require only 20% of that bandwidth (1.41 and 1.56 Gbps, respectively). Also, the peak server bandwidth for CPM and P2P is at the beginning of the experiment indicating that once the peers are seeded, both CPM and P2P are able to serve requests by transferring data from the peers for this workload. Figure 3 also shows the total instantaneous traffic coming from peers serving requests for both P2P and CPM. There is not much difference between time periods 3000 and 8000 minutes between CPM and P2P. However, when the load is higher (at the beginning and at the end), CPM results in as much as 33% fewer peer transfers. This, combined with only a slight increase in the server bandwidth, is indicative of the fact that CPM replaces these peer transfers with multicasts from the server.

This result shows that while CPM is successful in dynamically determining the least cost way of delivery, P2P does very well too for this workload. In fact, it is a great example of why P2P-based approaches are gaining popularity: when there is locality (same content being requested by multiple clients) in the request stream and there is enough gap between the requests, P2P schemes are able to easily alleviate a significant portion of the load from the server. Instead, they direct requests towards peers that have already requested the content. The question that then arises, however, is: what if the load on the system changes? We examine this question next.

E. Performance under higher load

Since we observe an improvement for CPM when the load is higher at the beginning and end of the week’s trace, we now study the performance of the various schemes under a heavier traffic load. One may envisage a future environment where not only do the users view the (current) VoD content, but also view all video (including “linear”) broadcast TV in a time-shifted,

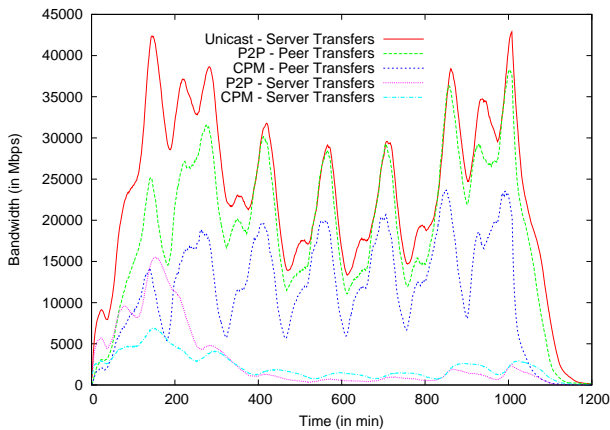


Fig. 4. Bandwidth required by different schemes to satisfy user requests following the real-world trace, but with requests arriving 10 times faster

on-demand manner. To model this, we scale up the arrival process. We use the same input dataset as Section IV-D, but have the requests arrive at 10 times higher rate. Instead of the 368K requests arriving over a week, they arrive in less than a day. We plot the results of this experiment in Figure 4.

As expected, Figure 4 shows that the server bandwidth for unicast goes up quite significantly (peak of 42.9 Gbps) with this higher arrival intensity. The plot also shows similar trends with P2P based schemes, with a lot more requests going to the server initially (to satisfy the requests when there are insufficient peers with that content). Subsequent requests are satisfied by the peers. Most importantly, however, the plot shows the advantage of using CPM over the other approaches, including P2P. Not only does CPM need lower server bandwidth (reduced by a factor of 3 compared to P2P and 1/8 compared to unicast), but also utilizes far fewer peer transfers ($\sim 1/2$ compared to P2P). This is because, when the request rate is much higher, there is more opportunity to multicast requests and peers are unable to satisfy as many requests (due to the increased request load on them).

The results in Figures 3 and 4 are significant because they indicate that CPM is able to identify and adapt to the best delivery mechanism possible, especially under high load, and that CPM works as well as, if not better than, P2P under light load. In essence, CPM can scale to a large client base more easily and successfully operate at different request regimes.

F. Sensitivity to system parameters

In this section, we look closely at different parameters that can affect the performance of the various delivery mechanisms. In order to be able to control the experiments and to make meaningful conclusions from them, we use a synthetic data set to drive these experiments. We experimented with a number of different parameters and scenarios as shown in Table I. Each of a 1000 clients played one 30 minute movie (chunk size of 30 secs., 7.5MB), had 20 Mbps download and 1 Mbps upload bandwidth reserved for the VoD service.

Based on the VoD data, our synthetic library consisted of 888 titles; 8 popular (chosen by 60% of the clients), 80 medium popular (chosen by 30% of clients), and 800 rare

Parameter	Default	Range
# clients	1000	-
Uplink BW	1 Mbps	500Kbps – 10 Mbps
Video bitrate	2 Mbps	-
Video length	30 min	-
Chunk length	30 sec	10 sec – 5 min
# videos (pop/med/rare)	8/80/800	8-50/80-500/800
Request arrival		
1st 50%	5 min	1 min – 5 hours
rest 50%	20 min	20 min – 5 hours
Pre-population	10 chunks	0 – all pop.

TABLE I
SIMULATION PARAMETERS

(chosen by remaining 10%). Each client chooses a movie within the category uniformly at random. This resulting popularity distribution across the requested videos was very similar to what we observed from the commercial VoD request traces.

By default, 50% of the clients request their videos in the first 5 minutes while the rest request uniformly over the first 20 minutes. CPM and P2P clients have local storage which they can use to cache chunks and in turn serve other peers. Clients store all chunks they download. Clients are prepopulated with chunks from the initial five minutes (10 chunks) of a popular movie (selected uniformly at random for each client). The P2P scheme is also allowed to stripe individual chunks from as many peers as possible.

All data points (except for single-run snapshots) are averages of five runs with different random seeds, and we have plotted the 95th% confidence intervals.

Server Capacity In Figure 5, we present results from an experiment in which we vary the server bandwidth (x -axis) and note the number of clients that are able to play out their video streams without buffer underrun (y -axis). All simulation parameters are set to the defaults in Table I. In each experiment, 1000 clients (the default number) start the simulation, but not all may finish if the server capacity is insufficient. The plot shows five different CPM variants, corresponding to the cases when CPM clients fetch one, two, ..., five chunks in parallel from other clients.

Unsurprisingly, the server demand for unicast increases linearly with the number of clients. For the default parameters, the server capacity required for multicast (1.07 Gbps) is less than P2P (1.29 Gbps) and also better than CPM (1.15 Gbps) when CPM clients only fetch one chunk at a time. However, as CPM is allowed to fetch multiple chunks (only from peers) simultaneously, the server capacity required reduces to 0.43 Gbps (with max. 4 simultaneous fetches). We reach a point of diminishing returns as the server capacity increases slightly with five simultaneous fetches for CPM.

Simultaneous fetches help CPM since it allows clients to fetch multiple chunks from other clients, building slack for subsequent fetches. The increased slack enables more peer transfers, and also builds up slack for when the client must go to the server. The server is able to use this increased slack to batch together more requests, eventually resulting in the lower server capacity requirement.

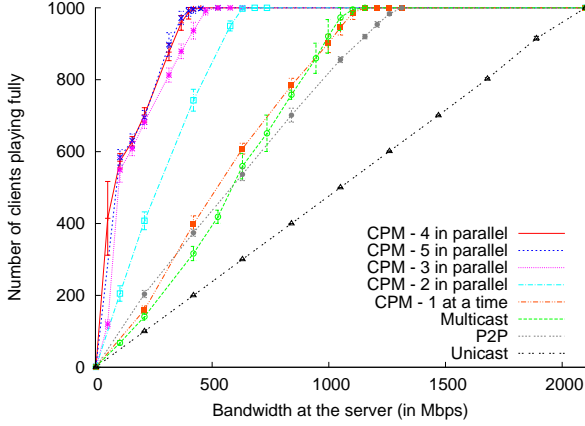


Fig. 5. Server bandwidth required for uninterrupted playback.

Sensitivity to Inter-arrival Time (IAT) In Figure 6, we vary the IAT distribution (x -axis) and plot the server bandwidth required to serve all 1000 clients. A pair (a,b) in the x -axis denotes the time over which the first 50% of the play requests were issued, followed by the time over which the rest of the clients issued their play requests. For instance, the x -axis value $(1,20)$ means that 50% of the clients requested a video during the first minute, and the remaining 50% requested a video over the first 20 minutes. The default value for other experiments corresponds to $x = (5,20)$ in this plot.

Unicast is insensitive to IAT when all the clients are active. As the request window increases to 60 minutes, the unicast server bandwidth required is halved since the load is halved (our videos are 30 minutes long). When the request window increases to 5 hours, requests are spread out and the bandwidth required at the server drops to 250 Mbps.

Multicast requires increasing server bandwidth as the request inter-arrival time increases, but is again able to reduce bandwidth required as clients finish playing before others start ($(60,60)$ and higher).

Requirements for P2P follow the trend of the unicast server, but require lower peak capacity because the P2P clients are able to help each other. P2P eventually outperforms multicast as more clients are able to help a sparse set of requests (all of which end up unbatched at the multicast server).

CPM is the most efficient protocol, and is relatively insensitive to demand variation since it is able to handle impulses ($(1/20)$, $(5/20)$) by multicasting, and sparse requests ($(60/60)$ and above) by peer transfers.

Changing Video Popularity CPM benefits from being able to multicast popular videos. To isolate this effect, we vary the popular set from 8 to 50, and the size of the medium popular set from 80 to 500. We do not vary any other parameters (including the number of rare movies, which are fixed at 800). As with other experiments, clients choose movies from the popular set with probability 60%, from the medium popular set with prob. 30%, and the rare set with prob. 10%.

Figure 7 shows the server bandwidth required for uninterrupted transfer to clients as the popularity is varied. As

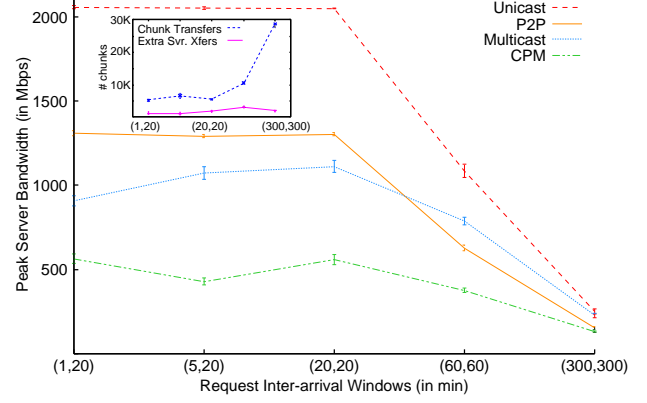


Fig. 6. Effect of request inter-arrival time.

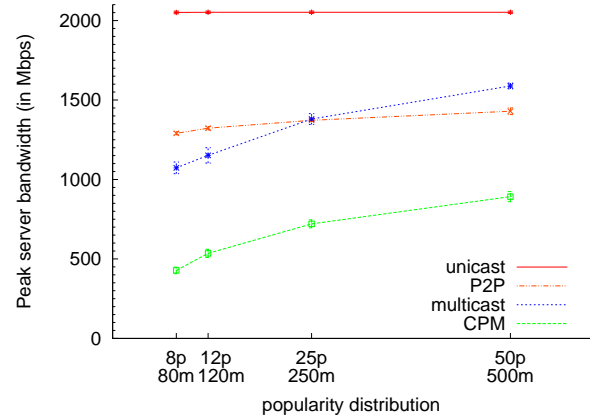


Fig. 7. Effect of video popularity. The plot depicts the server load, as the set of popular and occasional videos increases

expected, unicast is oblivious to popularity. Multicast requires increasing bandwidth as the size of the popular and medium sets increase. This is again expected since there is less opportunity for batching as client requests will form smaller batches.

The server bandwidth requirement for P2P scheme increases slightly with diffusing popularity. As the popular set increases, the number of clients that are pre-populated with a popular movie decreases, resulting in an increase in server load.

CPM, while consistently requiring lower server capacity than other schemes, requires increased server bandwidth as the size of the video set increases. Much like multicast, the CPM server is not able to aggregate requests as well, and like P2P, the client transfer is not as efficient.

Varying peer uplinks P2P transfers are eventually bottlenecked by the client's uplink. We have set our default uplink to 1 Mbps since this seems to be a commonly available rate in the upcoming roll-outs for broadband connectivity. Note that currently available uplink bandwidths are significantly lower, e.g., the median uplink bandwidth reported in recent studies range from 300–500 Kbps [12], [13].

Figure 8 shows how P2P systems might fare if future deployments provide much higher uplink capacities. If the uplink is increased to 2.5Mbps, CPM server requirement is only 25% less than P2P, and the P2P systems perform on par (within 12%) with CPM if the uplink bandwidth is increased

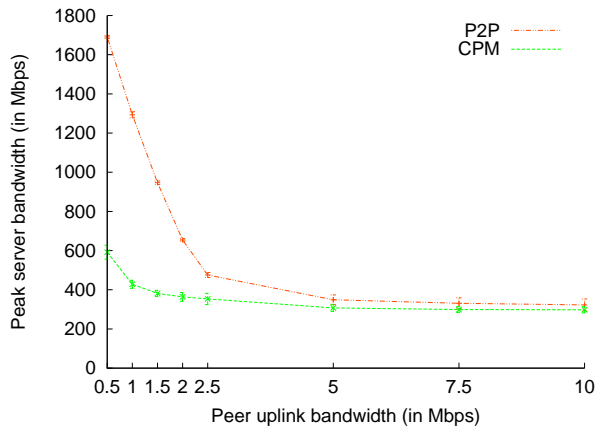


Fig. 8. Peak server bandwidth needed as peer uplink increases.

to 5 Mbps. There is marginal benefit beyond 5 Mbps.

We note that the critical parameter is not the value of the uplink bandwidth itself, but the ratio of stream bandwidth to uplink bandwidth. P2P systems do well when this ratio is higher than 1. CPM, on the other hand, does not require client uplink bandwidth to be on par with stream bandwidth to be effective for this workload.

Effect of Client Failures In this section, we quantify the effect of peer failures on CPM. We assume a fail-stop model where a peer that fails stops transferring or requesting data from that point onwards. Peers are modeled as failing independently and uniformly at random within the first 30 minutes of the experiment. We increase the number of peers that fail from 5% to 50% and measure how many peer transfers are affected by these failures and where the affected client goes to re-fetch the chunk.

% of clients failing	# of peer transfers	affected downloads	re-fetched from peers
5	7631.2	12	8.4
10	7274	21.6	15
25	6532.4	58.4	40.6
50	5395.6	107.8	68

TABLE II

EFFECT OF PEER FAILURES ON CPM. ALL THE ACTIVE CLIENTS PLAY THEIR VIDEOS FULLY, WITHOUT DISRUPTIONS

In all the five runs of the experiments, *all* the active clients completed playing their videos successfully. Table II shows that, of the relatively small number of affected downloads, nearly two-thirds of these failed transfers are recovered by clients fetching the chunk from another peer. The remaining third of the failed transfers are recovered by going to the server. To a limited extent, this suggests that CPM is resilient to churn among the clients.

V. RELATED WORK

Past approaches for VoD can be classified into three broad categories: server-based, edge-caching and peer-assisted. We discuss each of these in turn.

Server-based Approaches In order to overcome scalability limitations with unicasting videos, multiple variations based

on multicast [1], [14]–[16] have been proposed. For example, in Skyscraper and Pyramid multicasts [14], [15], multicast streams of popular videos are started at periodic intervals. With multicast, the number of streams from the server grows with the number of individual streams per VoD and is independent of the number of viewers. With VoD, however, videos requests arrive at different times and need not coincide with the start of a multicast stream. Hence techniques such as generalized batching or patching [17], [18] are used to allow the user to *catch-up* with an existing multicast. It has been shown that theoretically, the required bandwidth grows as the square root of the request arrival rate [19] and can be prohibitively expensive in practice [20].

Edge Caching Edge caching attempts to reduce the load on the VoD server by introducing caches in the edge of the network. When clients under the cache request for a popular video, the request is redirected to the cache rather than the video server. Since requests for popular videos comprise of much of the traffic, these caches reduce the number of requests going to the server. Entire videos or portions can be cached the first time they are requested or can be pre-populated. [2], [21] discuss techniques to cache prefixes of videos for efficient streaming. Furthermore, as shown in [22], these caches can be arranged as hierarchies such that they do not have to store the entire content. Instead, they can cooperatively cache the content. Such edge caches have been used successfully by commercial Content Distribution Networks like Akamai [23] and Limelight [24] to provide videos. In our setting, edge caching is equivalent to unicast since our video servers are already at the “edge” of the network.

Peer-assisted VoD P2P IPTv schemes use peer disk and bandwidth to implement VoD and live streaming services. Zigzag [25] and Nice [26] build a single overlay tree and multicast the video over this tree. Splitstream [27], CoopNet [28] and P2PCast [29] extend this idea to build multiple multicast trees for load balancing and resiliency against failures.

Swarming protocols [3], [8], [9] divide content into fixed sized chunks; peers form a random mesh and use a randomized pull technique to distribute the video. Swarming protocols are more robust than tree-based protocols since clients do not need to transfer chunks in sequence, and do not have designated tree “parents” from whom they must get data. Huang et al. [30] were the first to investigate the potential in the reduction of server bandwidth costs with careful design of peer-assisted prefetching policies. The use of swarming has also been proposed in [4]–[6], [31], [32]. Annappureddy et al. [4] arrange the peers in a mesh topology and utilize a central directory server. [5] uses a combination of BitTorrent and DHTs to locate chunks and [6] uses Bullet [33] to construct an overlay mesh. Tewari et al. [34] proposed swarming-based live streaming for limited upstream bandwidths. BASS [32] is a near VoD service that employs a hybrid approach (video server combined with peer assists). Unlike CPM, in BASS, the server bandwidth grows linearly with the number of users.

[35] proposes a scheme to push content proactively to peers (taking advantage of idle periods). This initial placement increases content availability and the efficient use of peer uplink bandwidths.

VI. CONCLUSIONS

We have presented CPM, an approach that augments server unicast and multicast with peer assists for providing a robust commercial video-on-demand service. CPM combines unicast, multicast, and peer-transfer in a seamless manner to obtain chunks of a requested video, whether they are available at servers in a service provider hub or at peers that have either been prepopulated or have previously viewed the content.

We view the strength of CPM to be its ability to adapt to a wide range of situations, which we have demonstrated in this paper. While both CPM and existing approaches, especially those based on P2P techniques, work equally well with today's workloads, CPM's ability to adapt results in significantly better performance when the prevailing conditions change. We showed that CPM combines the effectiveness of server multicast at high request arrival rates with the power of peer-to-peer download that excels at low to moderate arrival rates, to provide uninterrupted playout of requested content in a manner that is relatively independent of the popularity of the content. The data model that CPM is built around lends itself to easy implementation of traditional VoD features such as fast-forward and rewind, and for "richer" search-and-view approaches such as jumping to a tagged point, dynamic composition and other non-linear viewing. We view this as part of our future work.

REFERENCES

- [1] K. C. Almeroth and M. H. Ammar, "On the use of multicast delivery to provide a scalable and interactive video on demand service," *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 6, 1996.
- [2] Y. Guo, S. Sen, and D. Towsley, "Prefix caching assisted periodic broadcast for streaming popular videos," in *Proceedings of International Conference on Communications*, 2002.
- [3] X. Zhang, J. Liu, B. Li, and Y.-S.P.Yum, "Coolstreaming/donet : A data-driven overlay network for live media streaming," in *Proceedings of IEEE INFOCOM*, March 2005.
- [4] S. Annapureddy, S. Guha, C. Gkantsidis, D. Gunawardena, and P. Rodriguez, "Is high-quality VoD feasible using p2p swarming?" in *WWW '07: Proceedings of the 16th international conference on World Wide Web*, May 2007.
- [5] V. Janardhan and H. Schulzrinne, "Peer assisted vod for set-top box based ip network," in *Proceedings of Peer-to-Peer Streaming and IP-TV Workshop 2007*, Kyoto, Japan, August 2007.
- [6] N. Vratonjic, P. Gupta, N. Knezevic, D. Kostic, and A. Rowstron, "Enabling dvd-like features in p2p video-on-demand systems," in *Proceedings of Peer-to-Peer Streaming and IP-TV Workshop 2007*, Kyoto, Japan, August 2007.
- [7] B. Fenner, M. Handley, H. Holbrook, and I. Kouvelas, "Protocol independent multicast - sparse mode (pim-sm): Protocol specification (revised)," Internet proposed standard RFC 4601, August 2006.
- [8] N. Magharei and R. Rejaie, "PRIME: Peer-to-peer Receiver-driven MESH-based Streaming," in *Proceedings of IEEE INFOCOM 2007*, May 2007.
- [9] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. Mohr, "Chainsaw: Eliminating trees from overlay multicast," in *IPTPS 2005*, Ithaca, NY, USA, Feb 2005.
- [10] N. Bansal, D. Coppersmith, and M. Sviridenko, "Improved approximation algorithms for broadcast scheduling," in *Proceedings of the 17th ACM-SIAM symposium on Discrete algorithm*, 2006.
- [11] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon, "I tube, everybody tubes: Analyzing the world's largest user generated content video system," in *Proc. of ACM Internet Measurement Conference*, San Diego, CA, October 2007.
- [12] A. Haerberlen, M. Dischinger, K. P. Gummadi, and S. Saroiu, "Monarch: a tool to emulate transport protocol flow over the internet at large," in *IMC '06*, 2006.
- [13] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "Do incentives build robustness in bittorrent?" in *4th USENIX Symposium on Networked Systems Design and Implementation*, Cambridge, MA, 2007.
- [14] K. A. Hua and S. Sheu, "Skyscraper broadcasting: a new broadcasting scheme for metropolitan video-on-demand systems," in *Proceedings of ACM SIGCOMM*, 1997.
- [15] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling policies for an on-demand video server with batching," in *MULTIMEDIA '94: Proceedings of the second ACM international conference on Multimedia*. New York, NY, USA: ACM Press, 1994, pp. 15–23.
- [16] H. Ma and K. G. Shin, "Multicast video-on-demand services," *SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 1, pp. 31–43, 2002.
- [17] K. A. Hua, Y. Cai, and S. Sheu, "Patching: a multicast technique for true video-on-demand services," in *Proceedings of the 6th ACM International Conference on Multimedia*, 1998.
- [18] S. Sheu, K. A. Hua, and W. Tavanapong, "Chaining: A generalized batching technique for video-on-demand systems," in *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, Ottawa, Canada, June 1997, pp. 110–117.
- [19] L. Gao and D. F. Towsley, "Threshold-based multicast for continuous media delivery," *IEEE Transactions on Multimedia*, vol. 3, no. 4, pp. 405–414, 2001.
- [20] D. E. Smith, "IPTV Bandwidth Demand: Multicast and Channel Surfing," in *Proceedings of IEEE INFOCOM 2007*, Anchorage, AK, May 2007.
- [21] B. Wang, S. Sen, M. Adler, and D. Towsley, "Optimal proxy cache allocation for efficient streaming media distribution," *IEEE Transactions on Multimedia*, vol. 6, no. 2, pp. 366–374, 2004.
- [22] J. Ni and D. H. Tsang, "Large-scale cooperative caching and application-level multicast in multimedia content delivery networks," *IEEE Communications Magazine*, vol. 43, no. 5, pp. 98–105, 2005.
- [23] Akamai, "<http://www.akamai.com>."
- [24] "Limelight networks," <http://www.limelightnetworks.com>.
- [25] D. A. Tran, K. A. Hua, and T. T. Do, "Zigzag: A peer-to-peer architecture for media streaming," *IEEE Journal on Selected Areas in Communications, Special Issue on Service Overlay Networks*, vol. 22, no. 1, 2004.
- [26] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," in *Proceedings of ACM SIGCOMM*, August 2002.
- [27] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: High-bandwidth multicast in cooperative environments," in *Proceedings of ACM SOSP*, October 2003.
- [28] V. N. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai, "Coopnet: distributing streaming media content using cooperative networking," in *Proceedings of ACM NOSSDAV*, May 2002.
- [29] Y. Guo, K. Suh, J. Kurose, and D. Towsley, "P2cast: peer-to-peer patching scheme for vod service," in *WWW '03: Proceedings of the 12th international conference on World Wide Web*. New York, NY, USA: ACM Press, 2003, pp. 301–309.
- [30] C. Huang, J. Li, and K. W. Ross, "Can internet video-on-demand be profitable?" in *Proceedings of ACM SIGCOMM*, 2007.
- [31] A. Vlavianos, M. Iliofotou, and M. Faloutsos, "BiToS: Enhancing bittorrent for supporting streaming applications," in *9th IEEE Global Internet Symposium (GI2006)*, Barcelona, Spain, April 2006.
- [32] C. Dana, D. Li, D. Harrison, and C.-N. Chuah, "BASS: Bittorrent assisted streaming system for video-on-demand," in *7th IEEE Workshop on Multimedia Signal Processing*, 2005.
- [33] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High bandwidth data dissemination using an overlay mesh," in *Proceedings of ACM SOSP*, October 2003.
- [34] S. Tewari and L. Kleinrock, "Torrent assisted streaming system for video-on-demand," in *Proceedings of IEEE NIME Workshop, CCNC*, Las Vegas, January 2007.
- [35] K. Suh, C. Diot, J. Kurose, L. Massoulie, C. Neumann, D. Towsley, and M. Varvello, "Push-to-peer video-on-demand system: design and evaluation," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 9, 2007.