# Deploying Video-on-Demand Services on Cable Networks

Matthew S. Allen, Ben Y. Zhao and Rich Wolski
Department of Computer Science, U. C. Santa Barbara
{*msa, ravenben, rich*}@*cs.ucsb.edu*

***Abstract*— Efficient video-on-demand (VoD) is a highly desired service for media and telecom providers. VoD allows subscribers to view any item in a large media catalog nearly-instantaneously. However, systems that provide this services currently require large amounts of centralized resources and significant bandwidth to accommodate their subscribers. Hardware requirements become more substantial as the service providers increase the catalog size or number of subscribers. In this paper, we describe how cable companies can leverage deployed hardware in a peer-to-peer architecture to provide an efficient alternative We propose a distributed VoD system, and use real measurements from a deployed VoD system to evaluate different design decisions. Our results show that with minor changes, currently deployed cable infrastructures can support a video-on-demand system that scales to a large number of users and catalog size with low centralized resources.**

## I. Introduction

Video-on-Demand (VoD) has been the subject of intense interest both in the research and commercial sectors for a number of years. The reason for this popularity is twofold. On one hand, VoD represents an attractive service that is expected to draw a large number of subscribers if implemented and deployed well. On the other, VoD services pose a challenge due to the large storage size, high bandwidth, and connectivity persistence required for deployment. These challenges become particularly acute as the number of subscribers and the area of coverage increase.

Due to these high demands, most solutions for providing VoD services are distributed. Many solutions service multimedia requests using high I/O media servers scattered throughout the deployment area. In some solutions, these servers simply replicate the media catalog [5], [22], while others attempt to automatically cache popular data only [4], [13]. Unfortunately, these solutions scale poorly because an increase in subscribers must be met with the purchase of expensive multimedia servers. Peer-to-peer systems, like [3], [12], [16] address this scalability problem by utilizing the subscribers themselves to serve data. Thus, as the subscriber population grows, the number of service providers grow also.

One attractive environment to deploy a peer-to-peer VoD system is on the infrastructure used by U.S. cable television companies. This infrastructure is pervasive and the cable subscriber base is large, but in recent years competition in this industry has grown. Thus, there is significant incentive to deploy a scalable VoD service in this environment. Also, the infrastructure itself has two components that can facilitate the development of a peer-to-peer system. First, the physical network layer features a broadcast-based, ethernet-like network in the last mile. Second, the cable company deploys dedicated computers to each subscriber's home in the form of set-top boxes, which allow subscribers to access television services.

In this paper, we propose a method for deploying a peer-to-peer architecture over cable infrastructure to act as a proxy-cache for VoD data. Our approach uses set-top boxes within each coaxial neighborhood as peer-to-peer storage for caching content. Additionally, we exploit the broadcast capability of the coaxial network to cache data at a local peer as it is being viewed. To our knowledge, this is the first work on peer-to-peer systems that specifically targets cable infrastructure. We evaluate our system through simulation using the PowerInfo trace [22] of real VoD usage collected from a Chinese telcom over a number of months. This allows us to evaluate how a real VoD service would have performed over a variety of topologies and caching strategies.

In so doing, our work makes the following contributions.

- We show that simple caching methods produce significant load reduction on central VoD servers.
- We show that performance scales well with increases to neighborhood size, subscriber population, and catalog size.
- We show that our solution is feasible using the limitations of current technologies.

## II. Video-on-Demand for Cable Networks

Video-on-demand is the next logical step in Internet-based content delivery. Media companies such as cable (Cox, Comcast), satellite (DirecTV, Dish Networks), and movie rental (Blockbuster, NetFlix) are all investigating the feasibility of delivering digital content to the home with minimal overhead costs. Video-on-demand through existing cable or satellite links provides an easy deployment channel for next-generation content distribution. Existing providers such as Cox, Comcast and DirecTV are already deploying prototype video-on-demand as a value-added service.

Even as media companies experiment with video-on-demand, they are fundamentally limited by the data distribution model. Ideally, media companies would offer large selections of movie, television, and music on-demand to all users. Unfortunately, the resulting traffic would cripple the existing delivery infrastructure. Centralized media servers would become disk I/O bottlenecks as well as bandwidth bottlenecks,

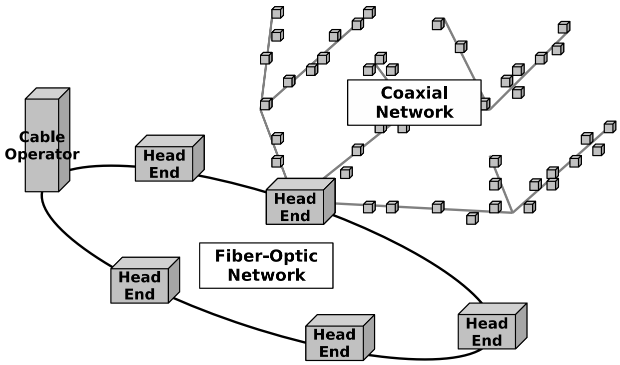



Fig. 1. Diagram of cable infrastructure

and poorly managed networks could become overloaded. Consequently, companies must limit content selection or limit accessibility to a subset of subscribers.

In particular, U.S. cable providers have a strong motivation to provide an extensive VoD service. Companies like NetFlix and DirectTV compete with the cable industry, and their successful deployment of a VoD system could bite into cable revenues. However, cable companies do have a pervasive hardware infrastructure in place, as well as a large subscriber base. In 1999, 99% of all homes in the U.S. reported owning a television, and 67% subscribed to a cable service [5]. Thus, there is tremendous incentive to deploy a scalable VoD system over this infrastructure.

The cable infrastructure is hierarchically organized into three components, which can be seen in figure 1. At the top of this hierarchy is the *cable operator*, which is the source of multimedia data. While separate services may be served from different geographic areas, we represent it here as a single source. The cable operator is connected to a collection of *headends*, which are the intermediate level of the hierarchy. Each headend is in turn connected to a set of *subscribers*, forming a *neighborhood* and competing the hierarchy.

The *Hybrid Fiber-Coax (HFC)* connects these components using two different physical networks. The cable operator and all headends are connected via a digital, switched *fiber-optic network*. This network provides high-capacity, low-interference, end-to-end connectivity between these two components. The headends are connected to the subscribers via a *coaxial* network. This is a legacy analog broadcast network that provided all cable services two decades ago. To broadcast television data to subscribers, the cable company generates data and sends it to each of the headends over the fiber network. Each headend then rebroadcasts all data it receives to its neighborhood on the coaxial network.

Coaxial networks have two important properties. First, they are broadcast based–any piece of data sent by the headend is seen by all subscribers. Likewise, any data sent by a subscriber can be seen by other subscribers. Second, coaxial networks are rate-limited and asymmetric. Current configurations focus on downstream traffic, supplying between 4.9 Gb/s and 6.6 Gb/s depending on the cable capacity. Of this, roughly 3.3 Gb/s are used for cable television, and the rest are used for other services. Upstream capacity receives a fixed and standardized allocation of approximately 215 Mb/s, which is used for IP cable modem traffic, set-top control signals, and VoIP data for the entire neighborhood. Because these properties have implications that negatively affect modern cable network usage, companies often push fiber closer to the subscriber's homes as budgets allow.

The final piece of cable architecture we will discuss is the *set-top box*, which is a specialized computer that provides cable services in the subscribers home. Its primary function is to pull data off the line, perform any necessary decoding, and display it on the TV. It also fields and handles requests for VoD, pay-per-view, or other other cable services, and may record programs if its a digital video recorder (DVR). Finally, it downloads targeted advertising and software upgrades and reports usage patterns in the background. While the subscriber can turn the set-top box "off"–thus deactivating the display and television signal–this device must remain on.

## III. Related Work

There has been a significant amount of work on the efficient distribution of video content on-demand over the network. Video data is very large, and the expected request rates in a VoD system are quite high. Video data ranges from 3.5 Mb/s to 8 Mb/s, depending on encoding quality, for normal-definition MPEG-2 video data. As a result, servers must both store a large amount of data and support a high I/O rate. The two main techniques that are used to address this problem are proxy caching and multicast. The former reduces server load by caching popular data strategically throughout the system. The later builds a multicast tree to attempt to distribute network load evenly throughout the system.

Much work in this area focuses on using high I/O servers to provide proxy caches for a multimedia distribution system. However, significant attention has also been dedicated to peer-to-peer solutions. These solutions exploit topology aware organization and quick replica location to produce a system that enables each peer to quickly locate and retrieve data. Many of these systems, like [2], [3], [10], [12] make use of structured peer-to-peer overlays [14], [19], [20], [24] to store specific data. Others, in [16], [17], attempt to form a less structured overlay for streaming media delivery.

### A. *Proxy Caching*

Proxy caches serve to reduce load on a central server and place data geographically closer to those accessing it. Systems are deployed by placing data caches strategically throughout a network. Clients wishing to access data first contact a nearby cache to request the data. If the data is popular, there is a good chance the request will score a cache hit and be returned immediately. Otherwise, the cache retrieves the data from a server, forwarding the response to the client and possibly caching it in the process. This solution is similar to web caching, which has been successful for companies like Akamai [1], [7].

Server based solutions place powerful machines throughout their deployment zone to provide caches. These machines are heavily optimized and expensive, and serve data to large numbers of customers. Research on these systems typically focus on sophisticated caching algorithms that achieve a high bit-to-hit ratio [4], [9], [13], [15], [25]. Fundamentally, servers are restricted by disk and network bandwidth. As a result, this solution scales poorly because if demand increases and the servers are at capacity, new servers must be purchased.

Peer-to-peer solutions attempt to address this scalability issue by staging proxies at nearby peers. Request are served by a peer if they can be served quickly, or by a server if they cannot. Because they lack a central location to collect viewing patterns and computer file popularity, populating the cache can be challenging. The systems described in [10], [11], which uses a peer-to-peer system to augment an existing proxy cache server, discusses this challenge. Also, because of the performance penalty from attempting to retrieve a cached item from a distant peer, systems must carefully address the issue of cache locality [2], [17].

### B. Multicast Trees

Multicast solutions make use of a multicast tree to distribute data to a large number of users with minimal server bandwidth. To accomplish this, clients requesting the same file coordinate to build an application level multicast tree. The server streams data to only a few nodes at the head of the tree. These nodes then retransmit the data to their children until the data propagates through the entire tree. Some methods, like [16], [23], [26], use an unstructured, gossip based protocol to multicast data. Others, like [3], [27], use a structured overlay to build a multicast tree.

Locality plays a significant role in the multicast solution, as low performance links can slow down every node below them in the tree. Works in [6], [16], [27] address this problem. Tree construction and maintenance are also challenging because failures in the interior nodes of the tree can be costly. Systems like [12], [21] build significant redundancy into each node in the multicast tree to combat this. Also, trees result in a greater savings in server bandwidth as they grow larger. Thus, systems like [6], [16] keep a playback cache at each node, so that if a new node joins the tree, it can catch up by downloading recent data from each peer's cache.

## IV. System Architecture

Our system deploys a VoD service across cable infrastructure using a proxy-cache architecture. We chose this method over multicast trees based on our analysis of VoD usage. Our architecture follows the cable topology to implement this system.

### A. Why Not Multicast

Multicast is an extremely popular research solution for providing scalable streaming services. Therefore, it is important to address in detail why we do not use multicast. Our reasoning is based on analysis of the PowerInfo trace of usage of a real VoD system [22], which we discuss in detail in section V-A. From this data, we determined two properties of video viewing patterns that undermine the effectiveness of multicast trees.

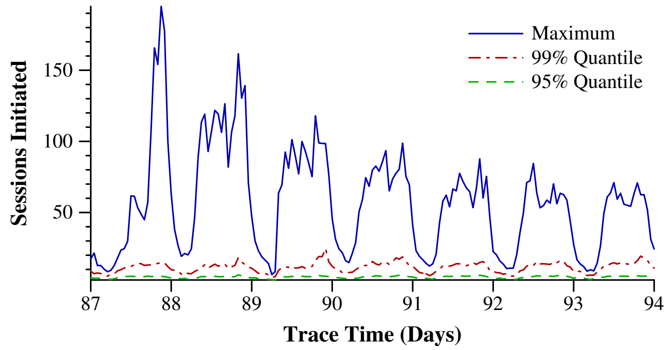



Fig. 2. Skew in file popularity during peak hours

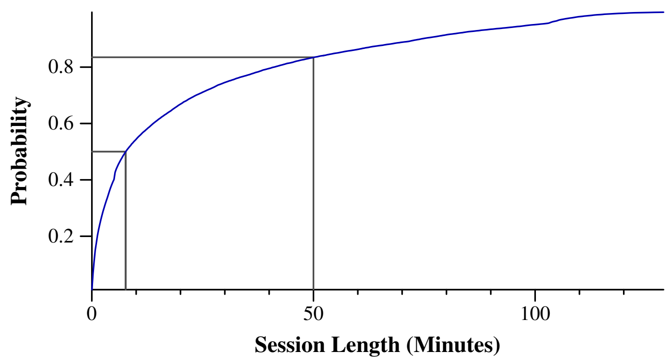



Fig. 3. CDF of session lengths demonstrating a high frequency of short sessions

First, the popularity of the files in this system is heavily skewed. We typically see a small number of extremely popular programs, and a very large number of unpopular ones. Figure 2 illustrates this skew. The solid line of this graph shows a running total the number of sessions initiated in the last 15 minutes for the most popular program during a seven day period. For the 99% quantile program, however, the number of accesses is down to around 13, and for the 95% quantile this number is down to 5. Multicast trees generate the most significant savings when many peers participate in the multicast tree. However, for the majority of programs in the catalog, it would be impossible to construct a large tree.

The second, and more significant, problem for multicast trees is that users in this VoD system have very short attention spans. Figure 3 shows the ECDF of the lengths of all sessions for the most popular file in the portion of the trace discussed in this section. For this 100 minute program, we see that 50% of the sessions last less than 8 minutes. Only 13% of all sessions surpass the half way mark. Modern multicast trees implement a variety of mechanisms to reduce the cost of peers leaving the tree mid-stream. However, our data shows that departures are quite severe, significantly complicating the maintenance of a multicast tree.

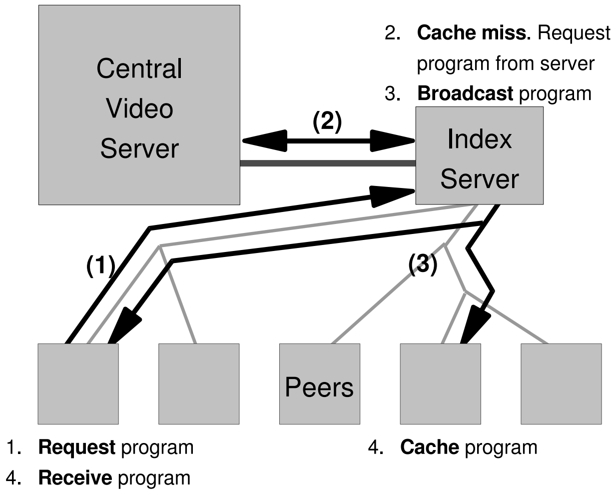



Fig. 4. Diagram of a cache miss

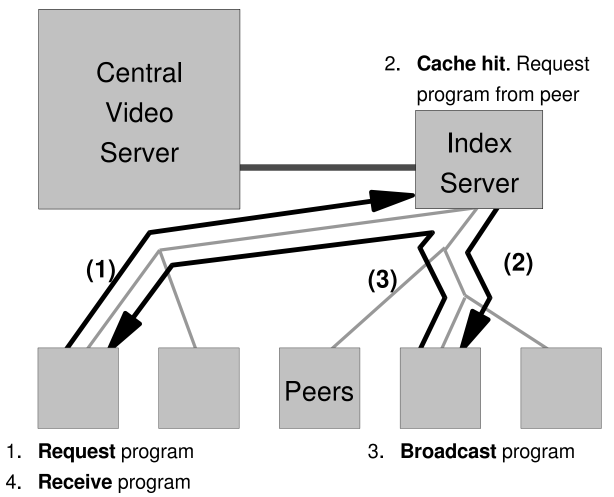



Fig. 5. Diagram of a cache hit

### *B. A Cooperative Caching Approach*

Due to our concerns regarding multicast trees, we propose a system based on a peer-to-peer proxy cache. Proxy cache solutions are less influenced by the skew in program popularity–a system caching the top 5% of files would be able to service a majority of the requests from cache. Also, proxy caches are not effected by high attrition rates like tree solutions are.

Our organization flows logically from the underlying physical architecture of the cable distribution network discussed in section II. The peers in our system are set-top boxes located at the edge of the network on the coaxial line. The peers in each neighborhood are organized into a cooperative cache by an *index server* placed at each headend. This server uses control signals to instruct peers to broadcast, store, or delete data as necessary to maintain the cache. The index server also monitors all requests in the neighborhood to calculate file popularity and populate the cache.

*1) Cache Implementation:* The placement of data in our system is dictated completely by the index servers. Programs are divided into 5 minute segments and distributed among a collection of peers. When the index server determines that a program should be in the cache, it locates a collection of peers to store the segments. It will also instruct peers to delete programs from their hard drives as the cache becomes full. Unlike many structured peer-to-peer systems, placement is not probabilistic. Instead, the index server places data to balance load, and keeps track of where each program is located.

The diagrams in figure 4 illustrates the interactions between the peers and index servers during a cache miss. Here, the flow begins in the lower left hand corner of the diagram with a request for a program segment (1). This request is received by the index server, which checks to see if the requested information is contained in the local cache. The index server sends a request to a central media server over the fiber-optic network (2), and broadcasts the newly received segment to the neighborhood (3). The requester then reads the data off the wire as it is broadcast (4). Also, if the index server has determined that the newly accessed program should be added to the cache, it may instruct another peer to read this same broadcast (4).

In the case of a hit, as shown in figure 5, the flow starts in the lower left (1). Instead of contacting a server, the index server locates the peer storing the segment and instructs it to broadcast (2). The peer broadcasts the segment (3), and it is received by the initial requester (4).

Data is transmitted at a rate of 8.06 Mb/s. This is the minimum rate necessary to sustain uninterrupted playback of a high quality MPEG-2 standard definition TV media stream. Many systems attempt to broadcast data faster than the playback rate because it allows users to then fast forward or skip ahead. We believe that, if network and server bandwidth are at a premium, these features should be provided using a more sophisticated encoding mechanism and not through inefficient and greedy network use. Fast-forward functionality can be implemented by sending an index of each segment in a program to subscribers and allowing jumps to predetermined points [8].

*2) Cache Strategies:* We look at two simple caching strategies that are implemented by the index servers. The most simple is a *Least Recently Used (LRU)* strategy. This strategy maintains a queue of each file sorted by when it was last accessed. When a file is accessed, it is located in the queue, updated, and moved to the front. If it is not in the cache already, it is added immediately. When the cache is full the program at the end of the queue is discarded.

The second strategy is a *Least Frequently Used (LFU)* strategy. To compute the cache contents, the index server keeps a history of all events that occur within the last $N$ hours (where $N$ is a parameter to the algorithm). It calculates the number of accesses for each program in this history. Items that are accessed the most frequently are stored in the cache, with ties being resolved using an LRU strategy.

*3) Set-Top Box Peers:* Peers in this system are provided by the set-top boxes that are distributed to all cable subscribers. Each peer in a network contributes a fixed amount of storage capacity to the distributed cache. The index server understands the total cache size to be the sum of the storage space contributed for each peer in the neighborhood. Set-top boxes are always on, which makes them particularly attractive for peer-to-peer systems. A major consideration for most such systems is "churn" [18], or the constant arrival and departure of peers. This is not an issue in our environment because of the characteristics of set-top boxes.

*4) Additional Requirements:* We make two significant additional requirements on the hardware in the HFC cable network. First, our experiments assume that some portion of the coaxial network be equipped with bidirectional amplifiers to allow all-to-all peer communication. Second, we expect that all set-top boxes run a peer-to-peer system and be equipped with both send and receive capabilities. Both these requirements demand a cost to the cable provider. Because we view the deployment of this infrastructure as an alternative to deploying fiber-optic cable closer to subscriber's homes, this cost is not unreasonable.

## V. Trace Driven Evaluation

Our simulations are based strictly on trace data collected from a real VoD system over a period of months. We use this trace to evaluate how well our caching architecture would have performed if it was serving the users of a real VoD service. Our simulation topology is designed to cover a variety of plausible cable network configurations. We evaluate our system in terms of the amount of VoD video data that must be served by centralized media server. Because of our trace data, we know exactly the load that these servers would have maintained given the level of encoding we chose for our experiments. This produces a realistic evaluation of how our system would have performed for a given topology configuration.

### *A. PowerInfo Trace Data*

The first component we will discuss is the PowerInfo VoD trace data, which we use to drive our simulation [22]. This data set captures every transaction that occurred in a deployed VoD system. The VoD service documented in this trace was provided in major cities in China by the China Telcom company. It was provided as part of an ADSL package to encourage users to purchase the service. This data set describes a single city where this service was deployed over a seven month period from May to December of 2004.

This data set contains 41,698 unique users who accessed a catalog of 8,278 unique programs. It contains over 20 million transaction records. Each of these records identifies the user, the program, and the length of the session. Because it was taken from a real system, the set demonstrates a variety of access patterns, user behaviors, and file popularity. These qualities would be difficult to model in a random and generic way. As a result, this data set provides us with the ability to benchmark a system as it would have performed in a real deployment. We feel this adds a tremendous amount of realism for our simulations.

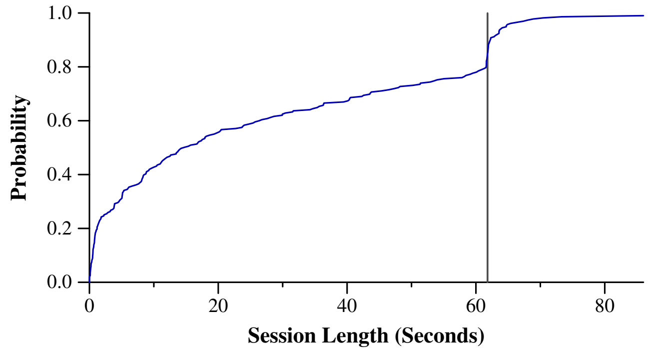



Fig. 6. CDF of session lengths demonstrating the approximate program length

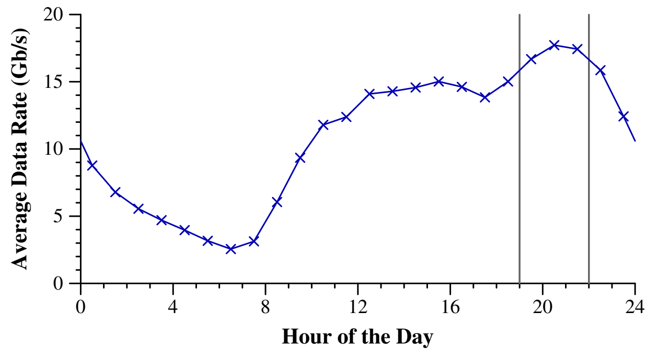



Fig. 7. Most popular hours for VoD usage

Unfortunately, one piece of information that was lacking from the data set was the length of each file. However, this information can be deduced from the file access patterns. While most users watch a program for a fraction of its length, a significant number do view the entire program. This pattern is shown in figure 6, which is an ECDF of all the accesses of a specific file in the system. In this graph, we see that a significant jump occurs at approximately 1 hour. This jump represents the fraction of users that watched the entire program, and is a pattern that is consistent with all program access patterns. We extrapolated the program lengths by manually inspecting the ECDFs for every program in the simulation for this pattern.

In this simulation, the most important metric is the peak data rate that the various architecture components must sustain. We know from the trace data that user activity reaches its climax between 7PM and 11PM in the evening. Figure 7 shows the average data rate that the VoD subscribers maintain for each hour of the day over the course of the trace. Based on this observation of the trace data, we focused on this three hour period when evaluating of our simulation performance.

For a collection of experiments, we did modify the trace data to increase both the number of users and the size of the program catalog. In order to minimize the extent of the changes, we strictly increased the number of agents to a multiple of the original number (double, triple, etc.). To increase the size of the catalog by a factor of $n$, we first create $n$ copies of every file in the trace. For each event in the trace, we substitute one of the $n$ copies of the original file at random. The method for increasing the number of users is similar. We create $n$ copies of each user, and for each event in the trace, we execute $n$ events–one for each copy–to the same file. In this case, we randomly change the start time between 1 and 60 seconds to eliminate problems caused by synchronous accesses. In this way, we can scale the number of agents in the trace without severely impacting the properties of the trace behavior.

### *B. Simulation Topology and Execution*

Upon initialization, the simulator associates users in the trace with subscribers in a neighborhood. The simulator places subscribers in neighborhoods uniformly at random. Neighborhood size is specified as a parameter to the simulation, and is chosen to reflect typical real world sized, which range between 100 and 1,000 subscribers. Peer placement is the same for each execution of the simulation with the same neighborhood size parameter. This is done so differences in the results of simulator executions are caused exclusively by algorithm performance and not user placement.

A discrete event simulation is dictated by each download event from the trace data. When an event occurs, the user who initiated the event locates the specified file in the simulated topology. This file will either be cached within the neighborhood by one of the peers, or it will be housed on a central server. In either case, the download consumes neighborhood bandwidth, and in the later case, it also consumes server bandwidth. The data rates sustained by the centralized servers and neighborhood networks for each hour of the day are updated with each event. These values are reported at the conclusion of the simulation.

### *C. Peer Restrictions*

Because set top boxes are distributed to all subscribers, they are optimized for low cost. As a result, their capabilities are extremely limited, which we account for in our simulation. Set-top boxes have limited disk space to contribute to a peer-to-peer cache. Current models commonly support hard drives of around 40 GB. We assume that set-top boxes will not be able to contribute more than 10 GB of these resources. Also, typical set top boxes cannot receive data on more than two logical "channels" of the coaxial line. This means, at worst, they can only receive two streams at once. We therefore limit each set top box so that it can only be active on two streams. The cache will trigger a miss if a segment is requested from a peer that has more than two active streams in either direction.

## VI. Experimental Results

In this section, we demonstrate the server load reduction achieved from deploying the system we described previously. We cover three topics in this section. First, we fully explore the performance of two caching strategies for a variety of realistic network configurations. Second, we demonstrate the feasibility

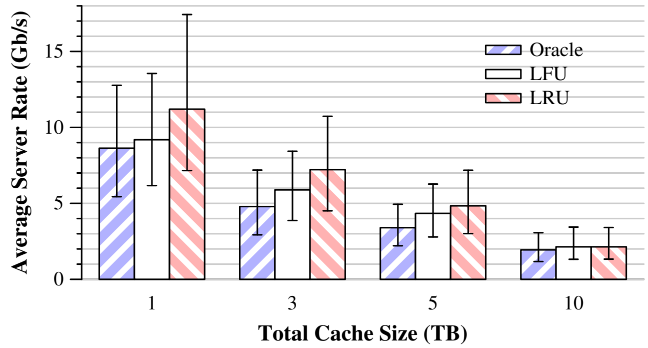



Fig. 8. Server load (neighborhood size fixed to 1,000 peers)

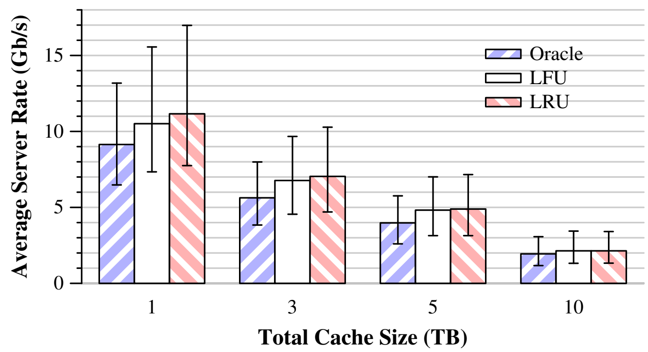



Fig. 9. Server load (per-peer storage fixed to 10 GB)

of our system given the real work constraints of our target architecture. Finally, we show that our system scales gracefully in the face of significant increased in user population and catalog size.

### *A. Effects of Caching*

The presence of the cooperative proxy cache in our VoD simulations has a significant performance impact, even with modest cache sizes or simple cache strategies. We present the effects of this cache for a variety of topology and cache strategies. Our analysis covers three caching strategies. **Least Recently Used (LRU)** and **Least Frequently Used (LFU)** were described in section IV-B.2. We benchmark both methods against an **Oracle** method, which caches the files that *will be* used the most frequently in the next three days. This final algorithm is impossible to implement, and is presented as an example of ideal cache performance.

First, we investigate the effects of total cache size on system performance. Figures 8 and 9 both show the average server load during peak hours for different cache sizes. The former changes the total cache size by keeping the neighborhood size fixed and varying the per-peer storage, while the later fixes the per-peer storage and varies neighborhood size. The error bars demarcate the 5% and 95% quantiles. With no cache, central servers must support 17 Gb/s. With 1 TB of total cache storage, this is reduced to around 10 GB/s, which is an impressive 35% improvement. However, a 10 TB cache size drops this load by 88% down to 2.1 GB/s.

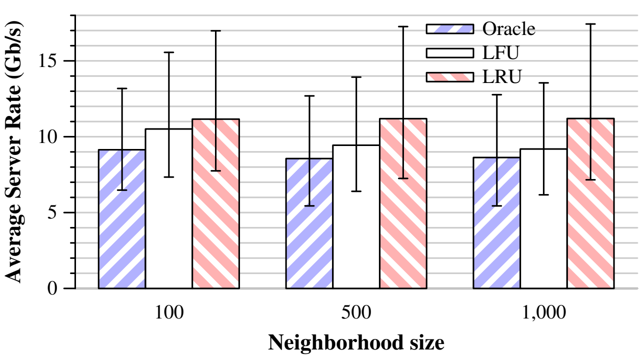



Fig. 10. Server load for neighborhoods of varying sizes

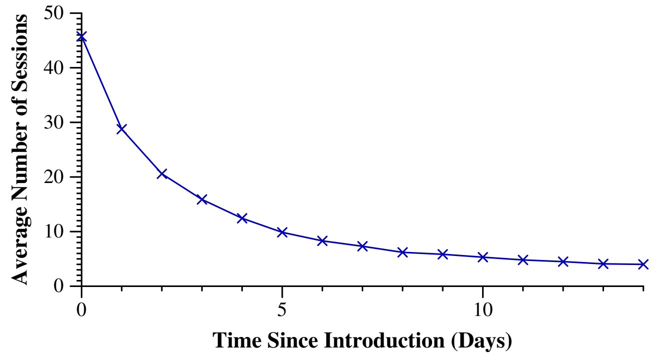



Fig. 12. Changes in file popularity in the days after introduction

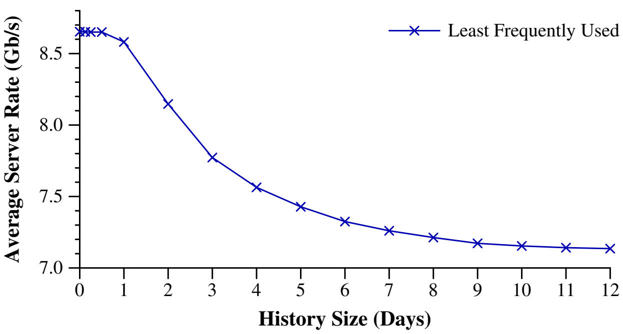



Fig. 11. Effects of history length on LFU strategy

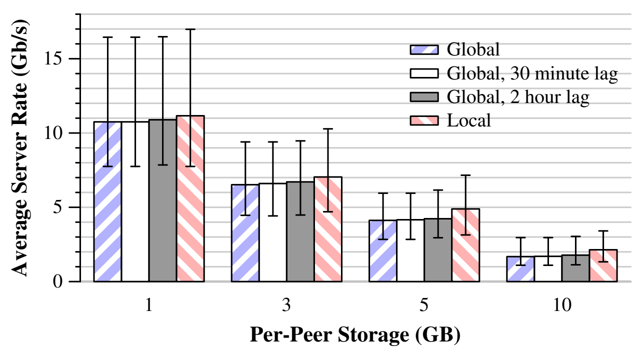



Fig. 13. Effects of using global popularity calculation of LFU strategy

These graphs also reveal the differences in the two caching strategies we investigate. These differences are most pronounced in small caches, which can only accommodate the most popular programs in the catalog. In these environments, an inaccurate cache will occasionally discard popular files due to a disproportionate lag between accesses or bursts of requests for a less popular file. As the cache grows large there is little, if any, difference between our two strategies. In these cases, the caches are large enough that they can accommodate all files that are accessed repeatedly. Nevertheless, we will point out that the LFU algorithm performs the same, if not better than, the LRU algorithm in all cases.

We investigate the differences between LRU and LFU for small cache sizes in figure 10. This graph shows the cache performance for different neighborhood sizes with a total cache size of 1 TB. As the network size increases, the performance of the LFU algorithm improves even though the total cache size stays fixed. This is because the LFU can make more accurate predictions of program popularity with more usage data. The LFU algorithm attempts to calculate the popularity of each file by looking at the number of requests that occur for the file in the neighborhood. The 1,000 node network will generate 10 times as much data for the LFU algorithm, resulting in better performance.

Just as the LFU strategy is affected by the amount of viewing information available, it is also affected by history size Figure 11 shows the cache performance for different history sizes of the LFU strategy in a 500 node, 2 TB neighborhood configuration. With a history size of 0, the LFU is simply an LRU strategy. As the history size increases up to 24 hours, we see little improvement over the LRU method, but after the 24 hour mark we begin to see significant savings with longer histories. However, this improvement tapers off with history sizes over one week. Although increasing history size increases the amount of data used for popularity prediction, long history sizes are in danger of including stale data. Figure 12 shows the number of concurrent accesses for the most popular programs in the days after its introduction. A week after introduction, programs are accessed 80% less often than the first day. Thus, calculating a files popularity including data that is a week old produces an inaccurate prediction of its current popularity.

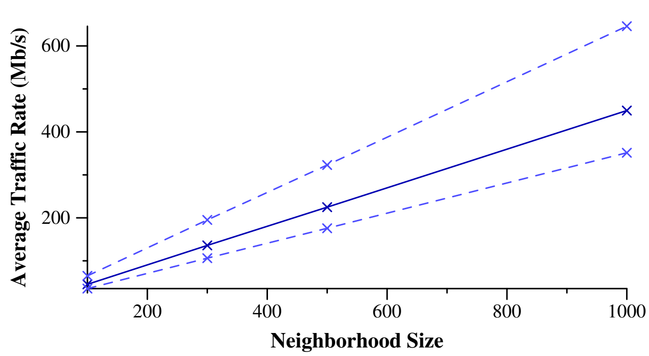



Fig. 14. Traffic on the coaxial network with varying neighborhood sizes

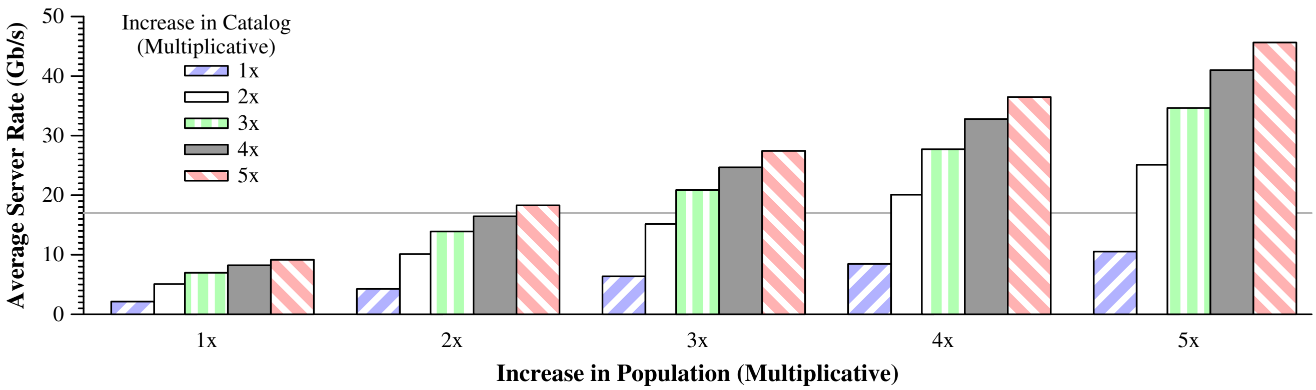
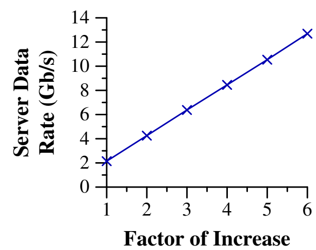


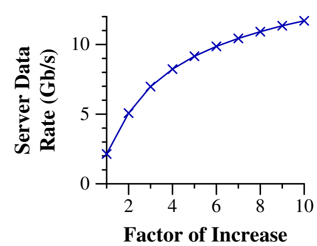

Fig. 15. Server load with increases in subscriber population and catalog size

| | | Increase in Catalog | | | | |
|---|---|---|---|---|---|---|
| | | 1x | 2x | 3x | 4x | 5x |
| Increase in Population | 1x | **2.14** | **5.07** | **6.98** | **8.23** | **9.16** |
| | 2x | **4.25** | **10.11** | **13.91** | **16.45** | 18.29 |
| | 3x | **6.38** | **15.15** | 20.87 | 24.67 | 27.44 |
| | 4x | **8.45** | 20.08 | 27.71 | 32.79 | 36.49 |
| | 5x | **10.54** | 25.11 | 34.65 | 41.01 | 45.64 |

(a) Server Load (Gb/s)




(b) Population Increase




(c) Catalog increase

Fig. 16. Detailed look at server load with increases in subscriber population and catalog size

One final way to increase the data available to the LFU algorithm is to use access data from peers outside the neighborhood. Figure 13 shows the cache performance if the neighborhood LFU algorithm is updated with usage data from all peers in the system. The bars on the left side show an LFU algorithm that uses complete global data to make every caching decision in the neighborhood proxy cache. The middle two bars show the performance if the local data is only augmented with global information in batches after a certain length of time has passed. The improvement from using global popularity information is noticeable, even if the global data is only incorporated periodically. However, the improvement in all cases is small. Although this technique could improve cache performance, it is unlikely to affect a significant change.

Concluding our exploration of caching strategies, we note that our caching mechanisms resulted in significant savings in server load. These savings are most pronounced in large cache sizes, which are a byproduct of large neighborhoods. Because our infrastructure is focused on reducing the high cost of pushing fiber towards the home, this result is important. Finally, while caching strategies do not have an overwhelming impact on server load reduction, they do have some. In particular, making LFU caching decisions with more program access data results in a performance improvement due to accurate popularity prediction.

## *B. Feasibility*

It is important to establish, through performance results, that the system we describe is feasible given the capabilities of the underlying architecture. In section V, we discuss the limitations we imposed on our infrastructure with great care. However, one issue we have left completely undiscussed is the strain placed on the neighborhood coaxial networks.

Figure 14 displays the average data rate sustained by the neighborhood network during the peak hours of the simulation. Notice the increase in traffic appears strictly linear with the increase in network size. For large neighborhoods, the results show a network load of 450 Mb/s on average, and 650 Mb/s in poor cases. This equates to less than 17% of the capacity of the coaxial line in extreme cases. This is a manageable level of traffic. Also, its important to note that because of the broadcast nature of the coaxial network, each file must consume the same bandwidth whether it is sent from a peer or the index server. This usage would not improve with a more centralized approach.

## *C. Scalability*

Our final experiments show that our caching strategy scales well with increases in the user population and program catalog. To perform these experiments, we modified the events in the original trace in the manner described in section V-A. We test the performance in an environment with 1,000 users and 10 GB of per-peer storage. In figure 15, each cluster of 5 bars shows the average server load as the user population is increased multiplicatively–from the original 41,700 users for the leftmost cluster, up to 2 million for the rightmost. Each of the 5 bars in a cluster represents a multiplicative increase in the size of the catalog. The light demarcation line running horizontally across the background at 17 Gb/s shows the server load required to support the VoD trace data with no cache. The information is duplicated in text form in table 16(a).

Figure 16(b) is provided to clarify the impact of population increase on the server load. This graph duplicates the leftmost bar in figure 15 with the rising diagonal stripe, and the leftmost column of table 16(a). It is clear from this graph that the relationship between server load and population size is linear–doubling the population size also doubles the server load. However, the percentage savings on central server load is fixed at 88% regardless of the increase in user population. This demonstrates the scalability of peer-to-peer solutions, where new subscribers provide more cache servers for the system.

Figure 16(b) shows a clearer picture of the effects of increasing the catalog size, which is seen also in the leftmost cluster of bars in figure 15 or the top row of table 16(a). Increasing the catalog size decreases the effectiveness of the cache by reducing the percentage of popular files the cache can store. This has the effect of increasing the number of popular files and reducing cache effectiveness. However, the impact of serving the most popular files is still the driving force in savings, which results in the diminishing impact of increasing the catalog size that we see here.

This data shows that, for significant increases in the scale of services provided, the server cost remains under what would have been necessary to supply the service with no cache. Cumulative increases in both the population and catalog are necessary to drive the server load over this threshold. These results demonstrates that our system is able to gracefully deal with increases in the scale of the services provided.

## VII. Conclusion

Video-on-demand is the future delivery model for a wide range of media content. In this paper, we use a complete trace of a deployed VoD prototype system with two major effects: to analyze the effectiveness of different solutions and to perform a realistic simulation. We focus on distributed caching schemes at the network edge in the form of localized set-top storage per cable subscriber. We investigate the critical issues, including caching algorithms, popularity prediction, medium contention, and scalability with larger user sizes and media libraries. Our results show that with existing wired cable infrastructures, cable companies can deploy large video-on-demand systems that efficiently use storage at the edge to drastically lower bandwidth costs for VoD servers.

## References


[1] The Akamai home page. http://www.akamai.com/.

[2] Annapureddy, S., Freedman, M. J., and Mazieres, D. Shark: Scaling file servers via cooperative caching. In *Networked Systems Design and Implementation (NSDI)* (Boston, MA, USA, May 2005).

[3] Castro, M., et al. Splitstream: High-bandwidth multicast in cooperative environments. In *Proc. of SOSP* (Lake Bolton, NY, Oct. 2003).

[4] Chen, S., Shen, B., Wee, S., and Zhang, X. Designs of high quality streaming proxy systems. In *Proc. of INFOCOM* (March 2004). .

[5] Ciciora, W., Farmer, J., Large, D., and Adams, M. *Modern Cable Television Technology*, 2nd ed. Elsevier Inc., 2004.

[6] Cui, Y., Li, B., and Nahrstedt, K. oStream: Asynchronous streaming multicast in application-layer overlay networks. *IEEE Journal on Selected Aread in Communications* (2004).

[7] Dilley, J., Maggs, B., Parikh, J., Prokop, H., Sitaraman, R., and Weihl, B. Globally distributed content delivery. 50–58.

[8] Fahmi, H., Latif, M., Sedigh-Ali, S., Ghafoor, A., Liu, P., and Hsu, L. Proxy servers for scalable interactive video support. 54–60.

[9] Guo, L., Chen, S., Xiao, Z., and Zhang, X. Disc: Dynamic interleaved segment caching for interactive streaming. In *Proc. of ICDCS* (Columbus, OH, USA, June 2005).

[10] Guo, L., Chen, S., and Zhang, X. Design and evaluation of a scalable and reliable p2p assisted proxy for onde-demand streaming media delivery. In *IEEE Transactions on Knowledge and Data Engineering* (2006), vol. 18, pp. 669–682.

[11] Ip, A. T., Liu, J., and Lui, J. C. COPACC: An architecture of cooperative proxy-client caching system for on-demand media streaming. *IEEE Transactions on Parallel and Distributed Systems* (2006).

[12] Kostic, D., et al. Bullet: high bandwidth data dissemination using an overlay mesh. In *Proc. of SOSP* (2003).

[13] Liu, J. *Web Content Delivery*. Kluwer Academic Publisher, 2005, ch. 1: Streaming Media Caching.

[14] Maymounkov, P., and Mazieres, D. Kademlia: A peer-to-peer information system based on the XOR metric. In *Proc. of IPTPS* (Cambridge, MA, March 2002).

[15] Miao, Z., and Ortega, A. Scalable proxy caching of video under storage constraints. *IEEE JSAC* (2002).

[16] Pai, V. S., et al. Chainsaw: Eliminating trees from overlay multicast. In *Proc. of IPTPS* (2005).

[17] Ramaswamy, L., Liu, L., and Zhang, J. Efficient formation of edge cache groups for dynamic content delivery. In *Proc. of ICDCS* (Lisboa, Portugal, July 2006).

[18] Rhea, S., Geels, D., Roscoe, T., and Kubiatowicz, J. Handling churn in a DHT. Tech. Rep. UCB//CSD-03-1299, University of California, Berkeley, December 2003.

[19] Rowstron, A., and Druschel, P. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. of Middleware* (November 2001).

[20] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of SIGCOMM* (August 2001).

[21] Tran, D., Hua, K., and Do, T. Zigzag: An efficient peer-to-peer scheme for media streaming. In *Proc. of INFOCOM* (April 2003).

[22] Yu, H., Zheng, D., Zhao, B. Y., and Zheng, W. Understanding user behavior in large-scale video-on-demand systems. In *Proc. of ACM EuroSys* (April 2006).

[23] Zhang, X., Liu, J., Li, B., and Yum, T.-S. P. CoolStreaming/DONet: A data-driven overlay network for peer-to-peer network for live media streaming. In *Proc. of INFOCOM* (Miami, FL, USA, March 2005).

[24] Zhao, B. Y., et al. Tapestry: A global-scale overlay for rapid service deployment. *IEEE JSAC 22*, 1 (January 2004).

[25] Zheng, C., Shen, G., and Li, S. Distributed prefetching scheme for random seek support in peer-to-peer streaming applications. In *ACM Workshop on Advances in Peer-to-Peer Multimedia Mtreaming* (2005).

[26] Zhou, M., and Liu, J. A hybrid overlay network for video-on-demand. In *IEEE International Conference on Communications* (2005).

[27] Zhuang, S. Q., et al. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proc. of NOSSDAV* (June 2001).