Balancing Throughput, Robustness, and In-Order Delivery in P2P VoD

Bin Fan*, David G. Andersen*, Michael Kaminsky†, Konstantina Papagiannaki †
*Carnegie Mellon University, †Intel Labs Pittsburgh

Abstract

Peer-to-peer has emerged in recent years as a promising approach to providing Video-on-Demand streaming. The design space, however, is vast and still not well understood—yet choosing the right approach is critical to system performance. This paper takes a fresh look at the p2p VoD design space using a simple analytical model that focuses on the allocation of uplink bandwidth resource for different chunks across peers. We describe a fundamental tradeoff that exists between system throughput, sequentiality of downloaded content and robustness to heterogeneous network conditions and node capacities, and we prove that *no system can achieve all three simultaneously*. Empirical results from Emulab confirm the analysis and show how one might implement efficient peer-to-peer VoD streaming with an appropriate balance of the tradeoff.

Categories and Subject Descriptors

D.2.2 [**Computer-Communication Networks**]: Distributed Systems —*Applications*; D.4 [**Performance of Systems**]: Modeling techniques

General Terms

Performance, Design, Load Balancing

Keywords

network, p2p, peer-to-peer, streaming, swarming

1. INTRODUCTION

Despite the wide success of peer-to-peer for bulk file transfers, and substantial interest in using peer-to-peer for streaming transfers such as Video-on-Demand (VoD) [9, 12, 4, 6, 24, 23], several fundamental questions remain unanswered about the design of streaming P2P systems. Today's systems, successful or otherwise, typically select one or a few streaming strategies from the (huge) design space, but without a concrete grounding in *why* those choices work well.

ACM CoNEXT 2010, Nov. 30–Dec. 3 2010, Philadelphia, USA.

The key differentiator between traditional P2P bulk transfers and streaming transfers is the requirement for approximately in-order delivery to the client. In this paper, we carefully examine the design space of techniques that attempt to meet this requirement. We demonstrate, both analytically and empirically, that the in-order retrieval requirement by playback creates a tension between three desirable attributes:

- *(T)hroughput*: the average completion time for peers to download the entire file;
- *(R)obustness*: the ability to withstand heterogeneous network conditions such as churn and node capacities;
- *(S)equentiality*: the degree to which the file is retrieved sequentially.

We prove that achieving all three attributes of this "TRS tradeoff" at the same time is impossible. We examine this tension in detail in Section 4.

To understand the essentials of this tradeoff, we define a *per-chunk capacity* model focusing on the allocation of uplink bandwidth resources among different chunks. Thus, we cast the tradeoff as a *load balancing* problem in resource allocation. The model makes it possible to analyze and compare many seemingly irrelevant techniques proposed or deployed in P2P VoD systems, such as *chunk prefetching* in order to have more chunks to share with others [1, 5, 12, 25, 28] or *biasing selection* towards neighbors who are less useful to other peers [1, 28]. Using our model, we show that all of these techniques succeed by balancing the per-chunk capacity. In Section 5, we apply the model to other approaches, such as *network coding*, *segment random*, and *hybrid schemes* that mix rarest random and sequential downloads.

The results from the modeling and analysis are not merely theoretical. Section 6 demonstrates, through implementation in a popular open-source BitTorrent client, that the schemes perform comparably in practice to the theoretical projections. An important result of this evaluation is showing that *many* previous designs (most of which were not directly compared against each other) act similarly to sustain throughput by slightly relaxing in-order delivery. The specific algorithmic details appear to matter less than where these approaches move to in the trade-off space.

In the following two sections, we give an overview of P2P VoD systems and their design goals. In Section 4 we elaborate on the TRS tradeoff analytically. Schemes that balance the tradeoff are examined in Section 5. Section 6

presents empirical results. Section 7 discusses related work, and we conclude in Section 8.

2. P2P BACKGROUND

In a traditional centralized solution for content distribution, large clusters of servers host the VoD content. These servers require substantial capital investment as well as administration, power, cooling, and maintenance. In response, recent attention has focused on using P2P systems to augment or replace these central VoD servers. P2P techniques seek to improve throughput for bulk downloads by having clients download pieces of the file in parallel from multiple sources (peers) who are simultaneously downloading the same file. The potential advantages of this approach include cost savings, scalability, and ease of deployment. Popular peer-to-peer systems such as Gnutella, Napster, and BitTorrent [8] helped P2P transfers become one of the major components of Internet traffic.

Peer-to-peer systems generally work as follows. Content providers insert into the system new files which are divided into *chunks* (8 to 256 KB in size commonly). The client software computes a hash, such as SHA-1, for each chunk in the file. It then creates a *recipe* for reconstructing the file that lists the chunk hashes and their size or offset. By downloading each chunk listed in the recipe and placing them in order, other clients can reassemble the original file. Many P2P systems also identify the entire file using the hash of the whole file contents.

Clients that want to download this file first download the recipe. The clients then use some discovery process to locate other peers in the system and to determine what chunks of the file each of those peers has. The P2P client software contacts some of these peers in parallel, identifies which chunks the peers have that it wants, and downloads some of those chunks. During the transfer, the peers typically exchange bitmaps with each other to keep each other up to date about which chunks they have finished downloading (and can therefore make available to other peers).

P2P file transfer and VoD systems differ from live streaming systems [10, 13, 29, 15, 3] in two significant ways. First, VoD and file-sharing have an advantage in that they can impose longer buffering times when needed, and they can buffer well in advance of playback when possible. In contrast, live streaming operates under more strict timing constraints, and content cannot be downloaded long before its playback time. Second, live streaming has the advantage that its clients are playing the content in rough synchrony, ensuring that most clients have the needed content at any given time. In practice, these constraints mean that solutions for live streaming look much more like multicast—creating high-bandwidth dissemination meshes, etc., that require very careful coordination between peers [10, 13]. In contrast, as we show in the later sections, relatively simple mechanisms work well for VoD-style workloads.

The design space of P2P VoD systems is quite large and has a number of dimensions, but the most important ones are *chunk selection*—which chunk to download next; *service selection*—which peer to download it from; *client selection*—which other peers to serve. In this work, we study and implement these three design choices and show that they have drastic effects on the downloading order and system throughput. Note that many other design decisions, orthogonal to the decisions we evaluate in this paper, also affect the performance of P2P systems. For example, some systems break the file into chunks of the same, static size [8]; others dynamically determine the chunk boundaries [20] using content-based methods such as Rabin fingerprinting. The original BitTorrent used a central "tracker" to permit peers to locate other peers downloading the same file; newer variants can use a distributed index structure such as a Distributed Hash Table (DHT) to accomplish the same goal. We discuss in Section 7 the relationship of significant related work on creating incentives for peers to upload [8, 18, 19], making efficient use of the network by finding nearby peers [27], and using strategies such as network coding to simplify data distribution [26]. In general, we believe that optimizations that we explore to these underlying mechanisms apply to a broad spectrum of P2P systems

3. VOD GOALS

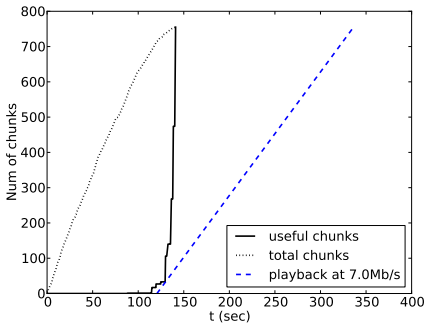
Different from conventional P2P file sharing systems designed to distribute files among a large number of users with **low total downloading time**, VoD streaming requires **short buffering time** because users wish to begin playing the downloaded content as soon as possible—certainly before the entire transfer completes. Throughout this paper, we define the buffering time as the minimal setup delay to ensure playback never runs out of useful chunks after it begins.[1]

With fixed playback rate encoded in the content (usually 400 Kbps–2 Mbps for standard definition video and 10 Mbps or more for high definition video), the buffering time depends on two underlying factors:

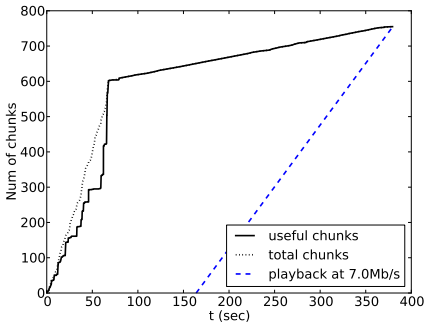
(1) **Throughput**: the number of bytes downloaded per second.
(2) **Sequentiality**: the order of chunk arrival. Because chunks of the video must be played back in order, their arrival pattern is critical (e.g., video playback cannot begin until at least the first chunk in the video has arrived).

Figure 1 illustrates the interplay between buffering time, throughput and sequentiality, calculated from real experiment traces. The dotted curve is the number of chunks downloaded by time t and the solid curve is the number of *useful chunks*, which is a subset of chunks in a contiguous sequence from the start of the file. The total time to download the video (determined by overall system throughput) is the rightmost point on the solid curve. The slope of the dashed "playback" line is the playback rate, and the x-intercept of this line is the buffering time.

[1] Note that if a user starts playback earlier, given the same useful chunk arrival pattern, the playback needs to pause later and wait until there are enough useful chunks. In that case, the total amount of buffering and re-buffering time will be the same as our buffering time.



(a) Rarest random (used for file sharing e.g. BitTorrent)



(b) Naive sequential (change BitTorrent client to download in order)

Figure 1: Buffering time by different chunk selection strategies.

From Figure 1(a), the rarest random strategy used by BitTorrent for file sharing shows high throughput (short downloading time) but the buffering time is almost as long as entire downloading because of the poor sequentiality. As a result the playback is bounded by the tangent line of the solid curve (useful chunks). Naive sequential (forcing BitTorrent to download chunks sequentially), as shown in Figure 1(b), has a desirable useful chunk arrival pattern since almost all downloaded chunks are useful. This strategy, however, is not scalable because the throughput collapses [2] and it takes a long time to finish downloading. The playback is bounded by the completion time.

Figure 1 highlights the importance of high throughput and good sequentiality for low buffering time. However it is also extremely important for a p2p system to withstand dynamic network conditions such as churn and bandwidth heterogeneity. Thus the third desirable property is robustness:

(3) **Robustness**: the ability to maintain high throughput in face of network conditions such as node failure, arrival/departure and heterogeneity of users' bandwidth.

[2]Most peers possessing the same collection of chunks fail to help each other, but only compete for the seed's uploading uploading resources and get a low downloading speed. With new and empty peers joining the system, starting to download and catching up with others quickly, the competition is getting even worse.

4. TRS TRADEOFF

In this section we present a fundamental tradeoff in P2P VoD streaming: *it is impossible to achieve maximal throughput, with purely sequential retrieval while being perfectly robust to variation at the same time*. We prove that a P2P VoD system can achieve at most two of the three attributes, but not all three at the same time. Three real-world examples help explain the design space:

- BitTorrent's pure rarest random strategy, illustrated in Figure 2(a), ensures high throughput and robustness, but its downloading is completely non-sequential (Figure 1(a)).
- Peers using pure sequential downloading among all neighbors in a greedy manner as illustrated in Figure 2(b) can retrieve content fully in order. This scheme, however, runs the risk of throughput collapse because the peers request chunks from the same sources which leads to a low utilization of system resources (Figure 1(b)).
- A system that sets up a linear "perfect cascade" of chunks from the source through each peer may achieve high throughput while delivering the file sequentially as in Figure 2(c), but is not robust—one slow peer will bottleneck the entire system.

In this section, we investigate the TRS tradeoff in detail.

4.1 Model Assumptions and Metrics

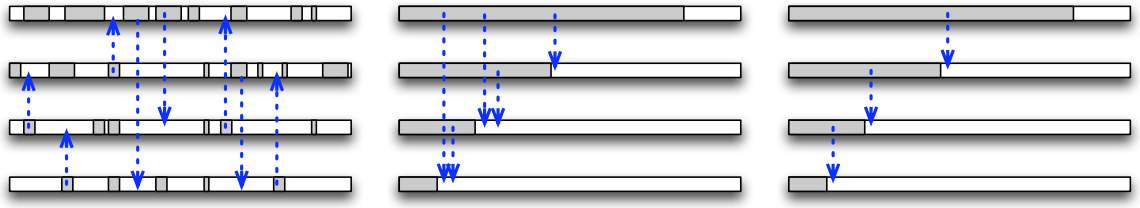
In this paper, we study a P2P VoD "swarm" where the served file is divided into M chunks of size b bytes each and peers arrive with a constant rate λ. To simplify the analysis, the following assumptions are made in our model:

- **Steady state:** The system operates in steady state, i.e., the rate of departures equals the rate of new arrivals and thus the population size of the swarm is stable.
- **Uplink capacity:** The bottleneck for disseminating the file is the peers' uplink bandwidth (not the downlinks) for most recent network environments [22]. All peers have the same uplink capacity U_p which is fully saturated [2]. We move the argument of heterogeneity into robustness.
- **Seeding capacity:** The uploading capacity of the seed in the swarm is U_s. Peers leave the system after downloading M chunks. The model can represent multiple seeds as a single seed with larger U_s.
- **Bandwidth allocation:** Seed and peers allocate their uplink bandwidth capacity uniformly among the chunks that they are serving (not necessarily all chunks that they have downloaded).

The notation used in this model is listed in Table 1. Given this model, we define metrics for throughput, robustness and sequentiality.

4.1.1 Throughput

There has been a significant amount of previous work studying the throughput of P2P systems [21, 16], but most of this literature focuses on deriving total throughput or average number of peers with an assumption of uniform chunk distribution. Such a simplification is reasonable in file sharing



(a) **Rarest Random**: peers download rare chunks, becoming useful to the others despite having few chunks

(b) **Naive Sequential**: peers with fewer chunks can get help from those with more but not vise versa, which makes chunks near the start have more sources.

(c) **Cascading**: peers only download from a subset of neighbors and a chain is formed to propagate chunks.

Figure 2: Each of the three schemes can achieve two attributes. Arrows represent data transfer

λ	arrival rate (peers/sec)
M	number of chunks (usually large)
b	chunk size (bytes)
r_i	average number of sources for chunk i
C_i	per-chunk capacity of chunk i (chunk/sec)
g_i	seed's fraction of uplink bandwidth dedicated to chunk i
f_i	average fraction of peer uplink bandwidth dedicated to chunk i
U_s	seed's uplink capacity (bytes/sec)
U_p	peer's uplink capacity (bytes/sec)
N	average number of peers in the system
T	average downloading time (sec)

Table 1: Formula Notation

systems where all chunks are equally important, but not true in streaming applications where peers prefer chunks close to their own playback position. Therefore in this paper, we propose a *per-chunk capacity model* focusing on the resource allocation of uplink capacities to different chunks.

We define *per-chunk capacity* C_i as the sum of the share of the uplink bandwidth allocated for chunk i from the seed and all other peers. C_i is the maximal rate at which the system can replicate chunk i. Obviously the sum of C_i cannot exceed the total uplink capacity in the system:

$$C_1 + C_2 + \ldots + C_M \leq U_s + NU_p. \tag{1}$$

From Eq. (1), having more peers (i.e., larger N) in the swarm can improve the per-chunk capacity.

To calculate a specific C_i, assume the seed allocates a fraction g_i ($\in [0,1]$) of its uplink bandwidth to chunk i, and on average a peer allocates f_i ($\in [0,1]$). g_i and f_i are determined by the streaming schemes (i.e., chunk selection, service selection and client selection as defined in Section 2). Following the assumption of bandwidth allocation, each peer allocates its uplink capacity uniformly to all chunks that have been downloaded and serving. Since the seed has all chunks, g_i equals $1/M$. The aggregate capacity for chunk i can be denoted by:

$$C_i = g_i U_s + N f_i U_p, \tag{2}$$

where $g_i U_s$ is the bandwidth contributed by the seed and $N f_i U_p$ is the total bandwidth contributed by all N peers in the system. Although in the analysis we assume all peers are homogeneous for simplicity, Section 6 includes an evaluation of systems with heterogeneous peers.

Given C_i, Theorem 1 relates the maximal arrival rate to the minimal per-chunk capacity in a system in steady state.

Theorem 1. *The maximal arrival rate λ_{max} that a system in steady state can sustain is bounded by the minimal per-chunk capacity divided by the chunk size b:*

$$\lambda_{max} \leq C_{min}/b = \min_{i=1\ldots M} \left\{C_i\right\}/b \tag{3}$$

Proof $A(t)$ and $D(t)$ are the number of peers that arrive into and depart from the system up to time t. $Q_i(t)$ is the number of copies of chunk i that have been replicated up to time t (including the copies already gone with peer departure and copies still in the system). $Q_i(T)$ is at least $D(T)$ because all peers that depart have already downloaded chunk i, and at most $A(T)$ because all peers that have joined the system need this chunk only once. Therefore,

$$D(T) \leq Q_i(T) \leq A(T), \qquad i = 1,2,\cdots,M.$$

By definition $A(T)/T$ is the arrival rate, $D(T)/T$ is the departure rate, and in steady state the arrival rate equals the departure rate, thus

$$\lambda = \frac{A(T)}{T} = \frac{Q_i(T)}{T} = \frac{D(T)}{T}. \tag{4}$$

$Q_i(T)/T$ is the rate of replicating chunk i, which is bounded by the per-chunk capacity C_i/b. Therefore $\lambda \leq C_i/b$. Since this inequality holds for $i = 1,2,\cdots M$, we can conclude $\lambda_{max} \leq C_{min}/b$. □

Discussion: Bottleneck chunk To sustain an arrival process with rate λ, the system must allocate uplink resources such

that even the bottleneck chunk k is assigned per-chunk capacity $b\lambda$. Since the seed contributes $g_k U_s$ and each peer contributes $f_k U_p$, according to Eq. (2) there must be N peers in the system to satisfy the arrival process where

$$N = \frac{b\lambda - g_k U_s}{f_k U_p}. \tag{5}$$

Applying Little's Law $N = \lambda T$ to Eq. (5), the average downloading time is:

$$T = \frac{b}{f_k U_p} - \frac{g_k U_s}{\lambda f_k U_p}. \tag{6}$$

There is a lower bound on T. Note that the minimal per-chunk capacity is always less or equal than the average per-chunk capacity, i.e. $C_{min} \leq (U_s + NU_p)/M$. Applying Theorem 1,

$$\lambda \leq \frac{U_s + NU_p}{bM}. \tag{7}$$

Substituting $N = \lambda T$ in Eq. (7), the lower bound of T is

$$T \geq \frac{bM}{U_p} - \frac{U_s}{\lambda U_p}. \tag{8}$$

Note that bM is the file size and bM/U_p is the time to download the file with the assistance of a peer dedicating all its uplink bandwidth.

4.1.2 Robustness

If peers have more available sources to download chunks,

- there is more flexibility to optimize delay or bandwidth in a dynamic or heterogeneous environment;
- the peer can handle churn better when connections are lost since it can quickly switch to other sources; but
- the benefits of additional peers for robustness diminish relatively quickly: having a few choices is important, but having a very large number is less so.

Therefore, we define our robustness metric based on the number of available sources. Let r_i be the number of available sources that each peer can download chunk i from. Note that r_i is not necessarily the number of peers actually having chunk i in the system. Using specific client and service selection policies that restrict a peer to only download from or upload to a specific subset of other peers, r_i can be smaller than the number of actual sources. Given $\bar{r} = (r_1 + r_2 + \cdots + r_M)/M$ as the average number of available sources over all chunks the robustness is defined as

$$R = 1 - p^{\bar{r}},$$

where $p \in (0,1)$ denotes the probability of a peer being "bad" (e.g., slow; failing; dropping the connection).[3] Intuitively, $1 - p^{\bar{r}}$ is the probability of having at least one good source to download from. Higher R indicates better robustness, but additional sources provide diminishing returns.

In a system in steady state with N peers, $\bar{r}$ is bounded. In steady state the probability for a randomly selected peer to have x chunks is $1/M$, for $x = 0, 1, \cdots M-1$. Thus the expected number of chunks that a random peer has downloaded is

$$\sum_{x=0}^{M-1} x \cdot \frac{1}{M} = \frac{M-1}{2}.$$

Then the sum of r_i over all chunks should be no more than the total number of chunks in the system:

$$\bar{r} = \frac{1}{M}\sum_{i=1}^{M} r_i \leq \frac{1}{M}\frac{M-1}{2}N \leq \frac{N}{2}.$$

In other words, the average number of sources over chunks is at most one half of the population size. Consequently, the robustness metric is also bounded:

$$R \leq 1 - p^{N/2}. \tag{9}$$

4.1.3 Sequentiality

To compare the sequentiality of different schemes, we first define *useful chunks* as the sequential chunks from the beginning of the file. Denote $U(x)$ as the fraction of useful chunks given x downloaded chunks. Based on $U(x)$, we define the sequentiality, S, of each scheme as the average "useful chunk fraction" over all chunks:

$$S = \frac{U(1) + U(2) + \cdots + U(M)}{M} \tag{10}$$

S is always between 0 and 1 because it is an average over fractions. A scheme that downloads chunks strictly in order has a sequentiality of 1 while a scheme that downloads the first chunk last has a sequentiality of $1/M$, which is close to 0 when M is large.

4.2 Three Basic Schemes

4.2.1 Rarest Random

Using the *Rarest Random* policy, each peer maintains bitmaps of all other peers indicating which chunks are available, then downloads the rarest chunk available from each neighbor. This scheme optimizes the even distribution of chunks, as illustrated in Figure 2(a). Below, we show how this distribution 1) produces a uniform per-chunk capacity C_i among chunks, 2) minimizes average downloading time as arrival rate λ scales, and 3) improves robustness by maximizing the number of sources for each chunk to download in the system.

The chunks are uniformly distributed among peers, thus the probability for a peer that has downloaded x chunks to have any particular chunk i is $\frac{x}{M}$. Therefore for this peer, chunk i obtains $1/x$ of the uplink bandwidth if it has been downloaded already (with probability $\frac{x}{M}$), and no share if it is still missing (with probability $1 - \frac{x}{M}$). As a result, on average

[3] The precise value of p is relatively unimportant; the robustness curve retains the same shape regardless.

the fraction of the uplink bandwidth allocated to each chunk i over all possible x is:

$$f_i = \sum_{x=0}^{M-1} \frac{1}{M}\left(\frac{x}{M}\cdot\frac{1}{x} + \left(1-\frac{x}{M}\right)\cdot 0\right) = \frac{M-1}{M^2} \approx \frac{1}{M} \quad \text{(M is large)}.$$

Substituting $g_i = f_i = 1/M$ to Eq. (2), the per-chunk capacity of each chunk by rarest random is uniform

$$C_{min}^{rr} = C_1^{rr} = \ldots = C_M^{rr} = \frac{U_s}{M} + \frac{NU_p}{M},$$

which is just $1/M$ (i.e. an equal share) of the total uplink resource. The average downloading time is given by Eq. (6):

$$T^{rr} = \frac{bM}{U_p} - \frac{U_s}{\lambda U_p}.$$

T^{rr} shows that *the rarest random scheme scales well*. When the arrival rate λ is very large (i.e., the swarm is very large), the average downloading time will still be at most $\frac{bM}{U_p}$ which is the average time to download the file with the assistance of one peer dedicating its full uplink. In fact T^{rr} is the lower bound given in Eq. (8) so *rarest random achieves perfect throughput*.

For robustness, let us first calculate the average number of sources for chunk i. The probability to have a chunk i given x downloaded chunks is $\frac{x}{M}$. In steady state, peers are downloading equally rapidly so the number of peers having x chunks ($x = 0, 1, \cdots, M-1$) is approximated by N/M, so on average the number of sources for chunk i is

$$r_i^{rr} = \sum_{x=0}^{M-1} \frac{N}{M}\frac{x}{M} \approx \frac{N}{2}.$$

Each chunk i has the same number of sources on average.

$$R^{rr} = 1 - p^{N/2}.$$

Recall the upper bound of R in Eq.(9), so *rarest random achieves the best robustness in steady state*.

For sequentiality, if a peer has downloaded x chunks, the expected number of useful chunks is:

$$E\Big[U(x)\Big] = E\Big[\text{num of useful chunks}\Big]/x$$

$$= \sum_{j=1}^{x} P\Big\{ \begin{array}{c}\text{num of}\\ \text{useful}\\ \text{chunks}\end{array} \geq j\Big\}/x = \frac{1}{x}\sum_{j=1}^{x}\frac{\binom{M-j}{x-j}}{\binom{M}{x}} = \frac{1}{M-x+1}$$

The metric of sequentiality according to Eq. (10) is:

$$S^{rr} = \frac{\sum_{x=1}^{M} E[U(x)]}{M} = \frac{\frac{1}{M} + \frac{1}{M-1} + \cdots + 1}{M} \approx \frac{\ln M}{M}$$

When M is large, S^{rr} approaches zero: *rarest random delivers content completely non-sequentially*.

4.2.2 Naive Sequential

Each peer downloads the next unrequested chunk in the order that those chunks would be played back, as illustrated in Figure 2(b). In naive sequential, peers download from random neighbors and serve all neighbors that send requests without selection or priority.

Using naive sequential, all chunks downloaded are useful. Also only peers with $i, i+1, \cdots, M$ chunks have chunk i. If the system is in steady state, the average number of peers with $0, 1, \cdots, M-1$ chunks is $\frac{N}{M}$. So the number of sources for chunk i is

$$r_i^{seq} = \sum_{x=0}^{i-1}\frac{N}{M}\cdot 0 + \sum_{x=i}^{M-1}\frac{N}{M}\cdot 1 = \left(1-\frac{i}{M}\right)N.$$

The chunk distribution is heavily biased toward early chunks. For example, almost all peers (except the empty ones) have chunk 1 and only the seed has chunk M (peers depart as soon as they obtain this chunk).

As a result, C_M is only contributed from the seed and we have the minimal per-chunk capacity:

$$b\lambda_{max}^{seq} \leq C_{min} \leq C_M = f_s^M U_s < U_s$$

C_M, the bottleneck per-chunk capacity, does not scale as the number of peers grows. This is a key difference from rarest random, which can handle any arrival rate because its bottleneck per-chunk capacity is proportional to the population size. With naive sequential, if the arrival rate is higher than the seed's uplink, the departure rate will be smaller than the arrival rate. And the population size will keep growing and *the system can never achieve steady state*.

Given r_i, the robustness is:

$$R^{seq} = 1 - p^{\sum_{i=1}^{M} r_i/M} = 1 - p^{N/2}$$

Naive sequential is as robust as rarest random[4].

Chunks are fetched in order, so all downloaded chunks are useful. *Naive sequential achieves the highest possible sequentiality*:

$$S^{seq} = 1$$

4.2.3 Cascading

In the previous two schemes, each peer can download from and upload to all other peers. Thus, the uplink capacity of each peer is shared by all chunks being downloaded. Downloading chunks strictly in order produces a skewed per-chunk capacity distribution.

To balance the per-chunk capacity, client selection or service selection policies can be applied to coordinate peers. For example, peers may form a chain to propagate the chunks from the source (seed), as illustrated in Figure 2(c). Peers only upload to other peers the portion of chunks that their

[4]This observation emphasizes that throughput and robustness are different metrics. Naive sequential obtains much lower throughput than rarest random, but the low throughput is robust.

immediate downstream neighbor needs. As a result, each peer uploads each chunk exactly once—hence, the per-chunk capacity is uniform. Cascading is also referred to as a "single-tree" scheme.

Peers depart from the cascade once they finish downloading the last chunk. Thus, only the seed can serve the last chunk. As a consequence, the per-chunk capacity can never exceed the seed's uplink. The per-chunk capacity is

$$C_{min} = C_1 = \ldots = C_M = \min\left\{U_s, \frac{NU_p}{M}\right\}.$$

If the seed is not the bottleneck, the downloading time is

$$T^{cas} = \frac{bM}{U_p},$$

which shows that *cascading almost achieves the perfect average downloading time*.

A peer cannot download from any neighbor, but only the direct upstream neighbor for all chunks. Therefore,

$$r_i^{cas} = 1,$$

and consequently,

$$R^{cas} = 1 - p.$$

Compared with naive sequential and rarest random, the average number of sources is much smaller for cascading. Therefore, *the robustness for cascading is the worst*.

Cascading peers fetch chunks strictly in order, so *the scheme is fully sequential*,

$$S^{cas} = 1.$$

4.3 Tradeoff Theorem

So far we have investigated three simple schemes, which are by no means the only choices in the design space. In fact other schemes based on these basic three—e.g. mixing rarest random with sequential or adaptive cascading by assisting slow peers—will be considered in Section 5. However these three schemes are the most representative because each scheme can achieve perfectly two of the attributes—throughput, robustness and sequentiality—but not all three.

In what follows, we prove that in fact the TRS tradeoff is fundamental and that no scheme can meet such an objective.

Theorem 2. *A P2P VoD system can not simultaneously maximize throughput, robustness and sequentiality.*

Proof Sketch Assume a scheme exists that achieves maximal throughput, robustness and sequentiality at the same time. The maximized throughput implies that $C_{min} = (U_s + NU_p)/M$ which is achieved if and only if $C_1 = C_2 = \ldots = C_M = (U_s + NU_p)/M$. Since this scheme also achieves pure sequentiality (all chunks are downloaded in order), any peer having chunk i also has chunk $1, 2, \cdots i-1$. If the system can also achieve perfect robustness ($N/2$ sources for each chunk), peers have to serve all chunks they have downloaded. Since each peer having i and j at the same time contribute the same amount to C_i and C_j and there are more peers having j, we know $C_j < C_i$, which contradicts the fact that throughput is maximized. □

Intuitively, this tradeoff is caused by the imbalanced resource allocation among different chunks due to the requirement of in-order delivery. In a system without delivery order requirement, disseminating chunks in random order can spread chunks uniformly throughout all peers and hence balance the uplink resource allocation. The sequential requirement breaks the balance because once chunk i is transferred to some peer, it always has to compete for uplink resource with chunk $1, 2, \cdots, i-1$, unless specific client and service selection polices are used.

In summary: **(1)** The minimum per-chunk capacity bounds the maximal arrival rate the system can sustain. To make the system scalable, the per-chunk capacity should scale with population size. **(2)** Rarest random maximizes throughput and robustness but has poor sequentiality. **(3)** Naive pure sequential downloading skews the chunk distribution and leads to a biased per-chunk capacity. Though it achieves perfect sequentiality, the system will nevertheless perform badly. **(4)** To keep in-order delivery scalable, client and service selection is necessary to force the peers to form a cascade. In this way, the system obtains an evenly distributed per-chunk capacity. But, this strategy has fewer available sources from which to download each chunk, making the system more vulnerable to churn and heterogeneity.

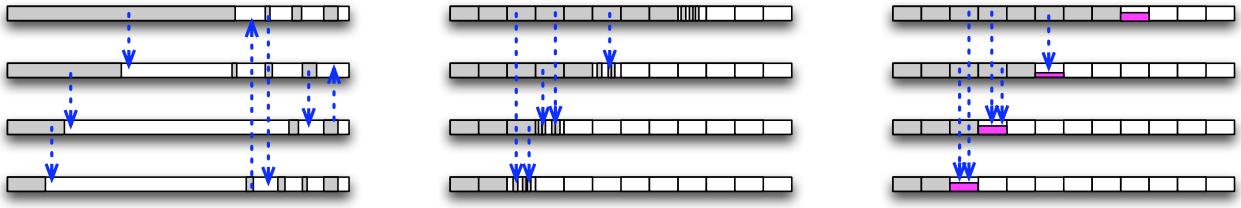
5. BALANCING THE TRADEOFF

One resolution to the tension between throughput, robustness and sequentiality is to aim to achieve acceptable, but not perfect, sequentiality and robustness, while maintaining high throughput. Doing so takes advantage of two factors: First, robustness provides diminishing returns; it may not be necessary to provide optimal robustness. Second, the most important metric for buffering time is the throughput of the sequential download. We show that reducing the sequentiality of the transfer can, perhaps counter-intuitively, *increase* the sequential throughput compared to a strictly sequential scheme. In this section, we study schemes that download chunks not strictly sequentially (see Figure 3).

5.1 Hybrid Strategy

One simple way to combine random and sequential download is to download a chunk according to naive sequential with probability $s \in (0,1)$, and according to rarest random with probability $1-s$. This *hybrid* chunk selection strategy downloads a large sequential chunk of the file plus a number of random chunks (as shown in Figure 3(a)).

We approximate this hybrid heuristic as follows: any time a peer has downloaded x chunks, sx of them are downloaded by the sequential component and placed consecutively from the beginning of the file; $(1-s)x$ of them are by rarest random and scattered randomly among the remaining $M - sx$ chunks.



(a) **Hybrid**: split bandwidth to fetch both sequential and rarest random chunks

(b) **Segment Random**: download rarest random chunks within sequential segments

(c) **Network Coding**: apply erasure coding for chunks in each segment

Figure 3: Schemes that balance the tradeoff. Arrows represent data transfer.

Thus, $P_{x,i}$, the probability of a peer to have chunk i, given it has x chunks in total, is

$$P_{x,i} = \begin{cases} 1 & \text{when } i \le sx \\ (x - sx)/(M - sx) & \text{when } i > sx \end{cases}. \quad (11)$$

If this peer has x chunks, each chunk gets $1/x$ share of the uplink bandwidth. The average fraction of uplink bandwidth allocated to chunk i over all peers is

$$f_i^{hybrid} = \frac{\sum_{x=1}^{M-1} P_{x,i} \cdot 1/x}{M} \quad (12)$$

The seed evenly distributes the capacity, i.e. $g_i = 1/M$. Substituting Eq. (12) to Eq. (2), the minimal per-chunk capacity C_i is

$$C_{min} = C_M = \frac{U_s}{M} + \frac{1}{M} \sum_{x=0}^{M-1} \frac{1-s}{M-sx} NU_p \quad (13)$$

The minimal per-chunk capacity C_{min} varies dramatically with s, the fraction of sequential downloading. C_{min} is a decreasing function of $s \in [0,1]$. It is maximized for $s = 0$ (i.e. pure rarest random) and minimized for $s = 1$ (i.e. pure naive sequential). When $0 < s < 1$ and M is large, C_{min} can be approximated by

$$C_{min} \approx \frac{U_s}{M} + \frac{(1-s)NU_p}{sM} \ln \frac{1}{1-s},$$

and T is calculated as

$$T^{hybrid} = \frac{s}{(1-s)\ln \frac{1}{1-s}} \left(\frac{bM}{U_p} - \frac{U_s}{\lambda U_p} \right).$$

With smaller s (i.e., more bandwidth dedicated to rarest random), T^{hybrid} is improved.

Since a peer always dedicates a fraction s for sequential fetch, the sequentiality will be at least s. Rarest random does not substantially improve sequentiality until the file is done,

$$S^{hybrid} \approx s$$

The fraction of sequential downloading, s, serves as a design knob: higher s improves sequentiality but may reduce the system throughput.

Discussion: Simple Bandwidth Division between sequential and random If a peer has downlink capacity d Mbps, and the playback rate of the movie is q Mbps ($d > q$), a naive streaming strategy would be downloading chunks sequentially at rate q and randomly at $d - q$. However this simple bandwidth division may not be the right choice. In terms of the preceding analysis, this is a hybrid scheme with $s = q/d$. When the downlink bandwidth d is close to the playback rate q (i.e., $s \to 1$), the system tends to behave just like naive sequential. As we know the naive sequential scheme is not scalable, this scheme may not be able to support sequential throughput larger than q Mbps.

Another option is adapting the ratio of sequential versus rarest random dynamically, e.g., increasing (decreasing) s if there are (not) enough useful chunks in the buffer as proposed in [28]. If this adaptive hybrid is used, once a number of peers run out of buffered useful chunks (e.g., due to congestion or a temporary seed failure) and tune s higher together. As a result downloading in the system will be more sequential and may reduce the system throughput. On the other hand, the decreasing throughput may, in turn, become a signal to encourage peers to be more greedy, further degrading throughput. Thus, this adaptive scheme has the risk of unstable throughput.

Choosing s dynamically in a distributed manner to optimize sequential throughput is an interesting research question.

5.2 Segment Random

The *Segment random* strategy groups all M chunks of the file into K segments, each of which consists of W chunks. Peers using this strategy download each segment strictly in order, but within each segment they fetch chunks according to rarest random. This is illustrated in Fig. 3(b).

To simplify the derivation of the per-chunk capacity, assume each peer allocates the same amount of uplink capacity to the segments being downloaded. Since segments are downloaded sequentially, in steady state there are $\frac{K-j+1}{K}N$ peers that have segment j (partly or completely) if the total population is N. In steady state, the rate of requesting and replicating each segment equals the arrival rate λ, so each peer having segment j receives $\frac{\lambda K}{(K-j+1)N}$ requests per time unit.

Assume each peer allocates bandwidth to a segment proportionally to the number of requests received for this segment.

The request rate for segment j is $\frac{\lambda K}{(K-j+1)N}$. Peers downloading the last segment have complete copies of segments $1,2,\cdots,K-1$ and a part of segment K. Segment K obtains the following fraction of the uplink bandwidth:

$$\frac{\frac{\lambda K}{(K-K+1)N}}{\sum_{j=1}^{K}\frac{\lambda K}{(K-j+1)N}} = \frac{1}{\sum_{j=1}^{K}\frac{1}{j}}$$

Only peers that are downloading the last segment contribute to its upload capacity. There are N/K such peers. Therefore, the average fraction of uplink allocated to chunks in the last segment is:

$$f_M^{sr} = \Big(\frac{1}{K}\frac{1}{1+\frac{1}{2}+\cdots+\frac{1}{K}}\Big)/W \approx \frac{1}{\ln K\cdot M}.$$

Therefore

$$C_M^{sr} = \frac{U_s}{M} + \frac{NU_p}{\ln KM}.$$

Similar to naive sequential, the bottleneck per-chunk capacity is C_M. But note that $f_M^{sr} = \frac{1}{\ln KM} > 0$: The downloading time may be very large, but the system can sustain an arbitrary arrival rate λ. The difference between segment random and naive sequential is that peers downloading chunks in the last segment can help upload this last segment, making the scheme more scalable than naive sequential. When $K = 1$ (i.e., the file consists of a single chunk), this scheme degenerates to rarest random; when $K = M$, the scheme becomes naive sequential.

For sequentiality, all chunks downloaded before the current segment (of size W) are useful, thus a peer with x chunks has at least $x-W$ useful chunks, i.e.

$$\frac{x-W}{x} < E[U(x)].$$

Therefore we can derive a lower bound of S^{sr}

$$S^{sr} > \frac{1}{M}\sum_{x=1}^{M}\frac{x-W}{x} = 1-\frac{W\ln M}{M}$$

When the segment size W is small or the number of chunks M is large, segment random can achieve a high sequentiality close to 1. A larger segment size increases throughput at the cost of sequentiality.

Discussion: Using Erasure Coding. Proposed network coding techniques for P2P VoD [1] divide files into segments. Within each segment, chunks are encoded so any received chunk is useful with high probability. On the other hand, peers cannot decode the content in a segment until it is complete.

Encoding creates more uploading opportunities for peers targeting the same segments and hence improves the throughput. The sequentiality is slightly worse than segment random because decoding requires W independent coded chunks in each segment.

Segment random and network coding do not fundamentally change the TRS when we consider the per segment capacity instead of per chunk capacity. The segment distribution is still biased as most peers have segments in the beginning and only the seed and a few peers can serve the last one and thus the per-segment capacity is still imbalanced.

5.3 Many More in the Space

Due to space constraints, we have only discussed hybrid and segment random in detail—two common techniques that abandon pure sequential downloading but aim at high sequential throughput. Note that there are other ways to add randomness to pure sequential transfers in order to smooth the per-chunk capacity. For example,

- A variation of hybrid picks the next chunk to download according to some heavy tail distribution in the chunk space [6, 4].
- A variation of segment random uses a sliding window of size W chunks. The beginning of the window is always the first missing chunk in playback order. Within the sliding window, rarest random [23] or hybrid chunk selection [28] can be used.

Other strategies trade robustness for throughput, e.g.:

- *Multi-cascading (multi-tree)* extends cascading (single-tree) so that each peer has multiple upstream/downstream neighbors to achieve good sequentiality while pursuing better robustness against heterogeneity and churn. The number of trees, however, is far smaller than $N/2$ (the average number of sources in rarest random) and cannot be as robust as rarest random.

In summary, *all of the schemes in the design space are subject to the TRS tradeoff*. Their goals are therefore necessarily the same: to smooth the per-chunk capacity to ensure that the systems scale well, by trading small amounts of robustness or sequentiality for increased overall throughput and to ensure that all peers have useful chunks to upload to other peers.

Discussion: Optimal Scheme? It is unclear that there is one optimal scheme because depending on the deployment scenario, robustness and sequentiality may be more or less important; the ratio of the peer's link capacity to the playback rate may vary, and so on. This may help explain the profusion of schemes to solve the problems identified herein. An interesting observation is that robustness has diminishing returns as the number of sources grows. In most cases, it makes more sense to give up perfect robustness to optimize throughput and sequentiality.

6. EVALUATION

The analysis in previous sections was necessarily simplified. In this section, to verify the analytical results and show the tradeoff in the real world, we evaluate the performance of several schemes empirically using Emulab.Due to space constraints, we have omitted PlanetLab results, which generally agree with the Emulab results below. Please see our tech report for an expanded evaluation.

6.1 Experiment Setup

Implementation: We implement the schemes in a Python implementation of the BitTorrent protocol, BitTornado 0.3.17. Our modifications consist of changes in the chunk, service and client selection algorithms. We disabled the tit-for-tat

policy because it is incompatible with schemes where peers download sequentially from neighbors they cannot help in return (e.g., naive sequential and cascading).

Topology: The experiments use one seed with up to 50 peers. We emulate a configuration similar to that of a P2P-assisted VoD service, with peers on the same cable head-end or DSLAM serving each other: the nodes are connected to the same switch via asymmetric links with capacity of 10 Mbps up, 20 Mbps down and 10 ms latency (this configuration is similar to a fast cable package today). In addition, we also run experiments with heterogeneous nodes (one third are significantly slower: 2 Mbps up and 5 Mbps down).

Workload: Each peer runs our BitTornado client to download a 197 MB video, which is divided into 754 chunks of size 256 KB.

Method: A seed is serving during the entire experiment. Peers join the swarm at a uniform spacing of λ peers per minute, for $\lambda = 2, 4, 6, 8, 12$. We choose an open-loop arrival to reflect more closely real P2P system arrivals. Once peers finish downloading the file, they leave the system without seeding to others.

6.2 TRS Tradeoff in Emulation

To illustrate the tradeoff, we compare the throughput of different schemes in Figure 4(a) and the sequentiality in Figure 4(b). We then change one third of the peers to be significantly slower (2 Mbps up and 5 Mbps down) as shown in Figure 4(c), in order to show the robustness of schemes against performance variation.

Tradeoff From Figure 4, we observe the tradeoff among throughput, sequentiality and robustness.

Rarest random achieves the highest throughput of all schemes, close to the uplink capacity (10 Mbps). Even with slow peers, the average throughput is still close to the average uplink capacity (7.33 Mbps). Throughput remains high as the arrival rate grows, which implies that this scheme scales well. The sequentiality, however, is less than 1%, which means the delivery is completely non-sequential and is not suitable for streaming applications.

Naive sequential achieves almost 90% sequentiality (see below for why). However, it does not maintain high throughput. The scheme is effective only when the arrival rate is very low ($\lambda = 2$ peers/min)—there are fewer peers online at the same time, so each gets more of the seed's capacity. As more peers join the swarm, the throughput collapses quickly.

Cascading helps peers download both sequentially ($> 99\%$) and quickly ($7-8$ Mbps). Unfortunately, the presence of slow peers cause the throughput to drop substantially compared to that of rarest random; the cascade is not robust to peer heterogeneity.

Hybrid, with 70% or 20% sequential downloading, balances rarest random and naive sequential. Although its sequentiality is worse than naive sequential, its throughput is drastically higher. In fact, at a high arrival rate, *both hybrid schemes achieve higher sequential throughput than naive sequential does*, while, of course, also attaining higher overall throughput. In other words, the throughput increase from adding modest amounts of rarest random is larger than the corresponding decrease in sequentiality. 20% sequential hybrid achieves throughput very close to that of pure rarest random, and its throughput remains high as the arrival rate increases. This throughput gain does come at the cost of substantially reduced sequentiality.

Segment Random also achieves a balance of rarest random and naive sequential. As the arrival rate grows, its performance begins to drop, showing its sequential bias.

Model Accuracy Figure 4 also shows the results predicted by our models.

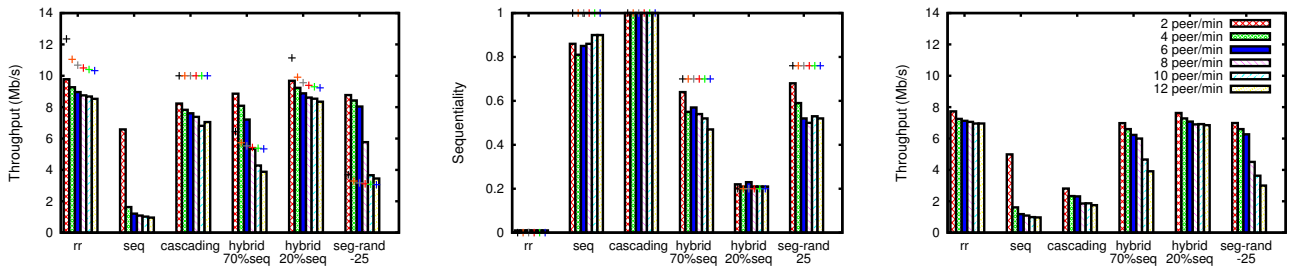
The model predicts the throughput trend as λ increases, but it overestimates the throughput of rarest random, cascading and hybrid-20%seq because it omits overhead from setting up connections, exchanging bitmaps and switching peers. The model underestimates the throughput of hybrid-70%seq and segment random particularly when the arrival rate is low. There are two reasons for this discrepancy: (1) We assume for simplicity that the seed will allocate its capacity uniformly across M chunks; in reality the allocation is biased because, as one of the few sources for those chunks, the seed receives more requests for them than for other chunks. In a system with only a few peers where the seed's contribution can dominate the per-chunk capacity, this bias substantially improves the throughput. (2) We could not allocate enough nodes to completely measure the steady state of slower schemes. When the arrival rate is high, schemes such as naive sequential, hybrid-70%seq and segment random require more nodes in steady state than we could obtain. Since the experiments end before the steady state, the throughput measured is higher than the real throughput in steady state.

The sequentiality of naive sequential, hybrid 70%seq and segment random are slightly lower than the model predicts. Peers in naive sequential, hybrid 70%seq and segment random download chunks from multiple sources. Chunks are requested sequentially, and most arrive in order, but some chunks are served by busy peers that take a long time to arrive. Those "late" chunks, while few, reduce the sequentiality. Only cascading achieves perfect sequentiality: each peer downloads only from its upstream peer, so chunks must arrive in the requested order.

6.3 Buffering Time

In Figure 5, we compare the buffering time of all schemes with and without slow peers.

The normal case where peers have the same uplink is shown in Figure 5(a). Cascading achieves the lowest buffering time, close to 0. This result makes sense because buffering time depends only on throughput and sequentiality, which cascading does very well at. The buffering time of naive sequential, due to the low and non-scalable throughput, is the highest, even though this scheme downloads chunks sequentially. Rarest

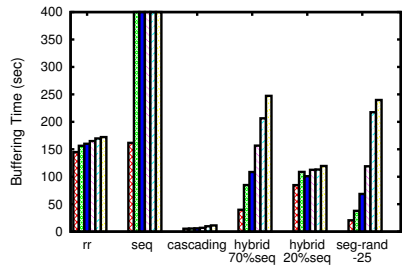


(a) Throughput: average downloading rate

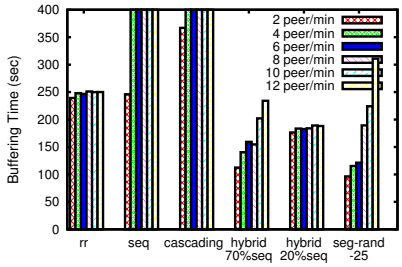
(b) Sequentiality:

(c) Robustness: average downloading rate under heterogeneity (1/3 nodes are slow)

Figure 4: Comparing schemes. "+" represents the model prediction for corresponding schemes.



(a) Buffering time with all peers having the same bandwidth



(b) Buffering time with one third slow peers

Figure 5: The buffering time of different schemes. Playback rate is 7 Mbps

random performs notably worse than hybrid and segment random especially when arrival rate is low due to the bad chunk arrival pattern. But when arrival rate is high, the buffering time becomes better than hybrid-70%seq and segment random because it has better throughput.

With a few slow peers (Figure 5(b)), the buffering time of cascading becomes dramatically larger due to the lack of ability to deal with heterogeneity. Hybrid and segment random outperform rarest random when arrival rate is low. With increasing arrival rate, a scheme suffers less in terms of buffering time if it has a larger rarest random component (which helps system scale better).

7. RELATED WORK

Inspired by the success of BitTorrent in file sharing, P2P VoD has been an extremely active area of research and deployment, with much research analyzing various design trade-offs. Many academic and commercial systems [9, 12, 6, 24, 1, 4, 23, 28] implement one or more of the policies discussed in this paper, including popular and effective variants of the hybrid strategy [4, 12, 6], variants of segment random [23, 24], and network coding [1]. Available research and documentation about these systems often demonstrates that the systems *do* work, but does not thoroughly explore the questions of *why* and *when* the systems work. Comparative studies such as Redcarpet [1], and studies of hybrid live streaming [31, 30] studied several different strategies, but to our knowledge, this paper is the first to comprehensively examine the fundamental limitations involved in designing streaming p2p systems, and how the major design choices for these systems (chunk selection and client/source selection) move through this space of tradeoffs.

Analytical models of P2P live streaming/VoD. In [14], a simple stochastic fluid model is developed to seek the characteristics of P2P live streaming systems such as buffering and playback lags. Recent works [31, 30] study the tradeoff between the startup delay and continuity in P2P live streaming systems and propose combining rarest random and sequential downloading. The startup delay in P2P VoD systems is also analyzed in [17] for rarest random, purely sequential retrieval and a proposed scheme similar to cascading.

Generally agreeing with this prior work, our analytical and empirical results focus more on the resource contention raised by the order of chunk downloading and a framework to understand why and how to balance the resource allocation. In addition, we complement our analytical model with a real p2p client implementation, thus deriving useful insight about the behavior of real systems; while we presented the material in this paper with the analysis first, the process of developing it depended strongly upon a back-and-forth between the analysis and the experimental results.

Incentives, caching, and prefetching. Encouraging [8, 18, 19]and enabling [11] clients to share more data plays an important role in the success of p2p systems. The interaction of existing incentive structures with VoD systems is an interesting and largely unaddressed question. BulletMedia [25] proposes to make peers proactively cache so as to help servers.

One encouraging result from our work is that *either* source or client selection can effectively increase sequential throughput, lending flexibility to designers trying to implement incentive schemes. Further, using the hybrid scheme that our work suggests performs best has the side effect of enabling tit-for-tat mechanisms to work with streaming workloads, because peers will again have chunks of data to share with each other.

Locality. Recent work shows that identifying and preferring peers that are "closer" in the network sense can improve throughput and reduce traffic on Internet backbones [27, 7]. As with incentives, above, we believe that these approaches are complimentary to our analysis, since they provide peers with a wide choice of sources from which to download, allowing locality techniques room to be effective.

Coding. Coding is a popular approach for transmitting data in many distributed systems, including video dissemination [26]. The key benefit of coding is to avoid explicit coordination (or at least weaken dependence on it) by simplifying scheduling and/or deal with unreliable transmissions. In this paper we discuss network coding as a generalization of segment random.

8. CONCLUSION

This paper presented a systematic examination of the design for peer-to-peer VoD through a combination of analysis and evaluation of a BitTorrent-derived prototype. The TRS tradeoff—a tension between throughput, sequentiality and robustness—is our main focus in this paper. The analysis of this tradeoff sheds light both on which transfer strategies are most effective, and *why* they are so. We apply the per-chunk capacity model to provide insights into schemes in the design space including rarest random, naive sequential, cascading, and two schemes that attempt to strike a balance in the tradeoff—hybrid and segment random. Empirical results from Emulab are used to confirm the analytical results.

Acknowledgments: The authors thank Vijay Vasudevan, Arvind Krishnamurthy, Michael Freedman and Scott Shenker for insightful and supportive feedback. This work was partially supported by NSF CAREER award CNS-0546551 and used equipment supported by MRI-0619525.

References

[1] Siddhartha Annapureddy et al. Is high-quality VoD feasible using P2P swarming? In *Proc. ACM WWW*, May 2007.

[2] A. R. Bharambe, C. Herley, and V. N. Padmanabhan. Analyzing and improving a bittorrent networks performance mechanism. In *Proc. IEEE INFOCOM*, Mar. 2006.

[3] Thomas Bonald et al. Epidemic live streaming: Optimal performance trade-offs. In *Proc. ACM SIGMETRICS*, June 2008.

[4] N. Carlsson and D. L. Eager. Peer-assisted on-demand streaming of stored media using BitTorrent-like protocols. In *Proc. IFIP/TC6 Networking*, pages 214–219, May 2007.

[5] Bin Cheng et al. A measurement study of a peer-to-peer video-on-demand system. In *IPTPS'07*, 2007.

[6] Yung Ryn Choe et al. Improving VoD server efficiency with BitTorrent. In *ACM Multimedia*, Sept. 2007.

[7] D. R. Choffnes and F. E. Bustamante. Taming the Torrent: A practical approach to reducing cross-ISP traffic in P2P systems. In *Proc. ACM SIGCOMM*, Aug. 2008.

[8] B. Cohen. Incentives build robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer Systems*, June 2003.

[9] C. Dana et al. BASS: BitTorrent assisted streaming system for video-on-demand. In *MMSP*, 2005.

[10] Y. hua Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *Proc. ACM SIGMETRICS*, pages 1–12, June 2000.

[11] C. Huang, J. Li, and K. W. Ross. Can Internet video-on-demand be profitable? In *Proc. ACM SIGCOMM*, Aug. 2007.

[12] Yan Huang et al. Challenges, design and analysis of a large-scale P2P-VoD system. In *Proc. ACM SIGCOMM*, Aug. 2008.

[13] Dejan Kostic et al. Bullet: High bandwidth data dissemination using an overlay mesh. In *10th SOSP*, Oct. 2003.

[14] R. Kumar, Y. Liu, and K. Ross. Stochastic fluid theory for P2P streaming systems. In *Proc. IEEE INFOCOM*, Mar. 2007.

[15] Shao Liu et al. Performance bounds for peer-assisted live streaming. In *Proc. ACM SIGMETRICS*, June 2008.

[16] L. Massoulie and M. Vojnovic. Coupon Replication Systems. In *Proc. ACM SIGMETRICS*, Oct. 2005.

[17] Nadim Parvez et al. Analysis of BitTorrent-like protocols for on-demand stored media streaming. In *Proc. ACM SIGMETRICS*, June 2008.

[18] Michael Piatek et al. Do incentives build robustness in BitTorrent? In *Proc. 4th USENIX NSDI*, Apr. 2007.

[19] Michael Piatek et al. One hop reputations for peer to peer file sharing workloads. In *Proc. 5th USENIX NSDI*, Apr. 2008.

[20] H. Pucha, D. G. Andersen, and M. Kaminsky. Exploiting similarity for multi-source downloads using file handprints. In *Proc. 4th USENIX NSDI*, Apr. 2007.

[21] D. Qiu and R. Srikant. Modeling and performance analysis of BitTorrent-like peer-to-peer networks. In *Proc. ACM SIGCOMM*, Aug. 2004.

[22] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Multimedia Computing and Networking 2002 (MMCN 02)*, Jan. 2002.

[23] P. Shah and J.-F. Paris. Peer-to-peer multimedia streaming using bittorrent. In *IPCCC 2007*, pages 340–347, Apr. 2007.

[24] A. Vlavianos, M. Iliofotou, and M. Faloutsos. BiToS: Enhancing BitTorrent for supporting streaming applications. In *9th IEEE Global Internet Symposium 2006*, 2006.

[25] Nevena Vratonijic et al. Enabling DVD-like features in P2P video-on-demand systems. In *ACM P2P-TV Workshop*, 2007.

[26] M. Wang and B. Li. r^2: Random push with random network coding in live peer-to-peer streaming. *IEEE JSAC*, Dec. 2007.

[27] Haiyong Xie et al. P4P: Portal for P2P Applications. In *Proc. ACM SIGCOMM*, Aug. 2008.

[28] Xiaoyuan Yang et al. Kangaroo: Video seeking in P2P systems. In *IPTPS2009*, 2009.

[29] X. Zhang et al. Coolstreaming/donet: A data-driven overlay network for efficient live media streaming. In *Proc. IEEE INFOCOM*, Mar. 2005.

[30] B. Q. Zhao, J. C. Lui, and D.-M. Chiu. Exploring the Optimal Chunk Selection Policy for Data-Driven P2P Streaming. In *The 9th International Conference on Peer-to-Peer Computing*, 2009.

[31] Y. Zhou, D.-M. Chiu, and J. C. S. Lui. A simple model for analyzing P2P streaming protocols. In *IEEE ICNP*, Oct. 2007.