Analysis of BitTorrent-like Protocols for On-Demand Stored Media Streaming

K.N. Parvez C. Williamson
University of Calgary
parvez,carey@cpsc.ucalgary.ca

Anirban Mahanti
IIT Delhi
mahanti@cse.iitd.ernet.in

Niklas Carlsson
University of Saskatchewan
carlsson@cs.usask.ca

ABSTRACT

This paper develops analytic models that characterize the behavior of on-demand stored media content delivery using BitTorrent-like protocols. The models capture the effects of different piece selection policies, including Rarest-First and two variants of In-Order. Our models provide insight into transient and steady-state system behavior, and help explain the sluggishness of the system with strict In-Order streaming. We use the models to compare different retrieval policies across a wide range of system parameters, including peer arrival rate, upload/download bandwidth, and seed residence time. We also provide quantitative results on the startup delays and retrieval times for streaming media delivery. Our results provide insights into the optimal design of peer-to-peer networks for on-demand media streaming.

Categories and Subject Descriptors

C.2.2 [**Computer-Communications Networks**]: Network Protocols; C.4 [**Computer Systems Organization**]: Performance of Systems

General Terms

Performance Analysis, Modeling, Simulation

Keywords

Peer-to-Peer Systems, BitTorrent, On-Demand Streaming

1. INTRODUCTION

Peer-to-peer (P2P) networks offer a promising approach for Internet-based media streaming. P2P networks are autonomous systems with the advantages of self-organization and self-adaptation. P2P solutions can enable efficient and scalable media streaming, as long as they can meet the sequential playback demands of *media streaming* applications, which differ from those of *file downloading*, for which P2P networks were originally created.

Recently, P2P networks have been used successfully for *live* media streaming, but the P2P paradigm is also applicable for the more difficult case of (near) on-demand streaming of *stored* media, which has received little attention. The two scenarios share several common challenges, including the sequential playback demands of large media objects, the geographic diversity of heterogeneous receivers, and the dynamic churn of the media streaming population.

On-demand streaming of stored media files differs in subtle but important ways from live media streaming. First, live streaming typically involves only a *single* streaming source, whereas stored media streaming can involve *many* providers of content. Second, the live streaming case allows peers to join at any time (i.e., mid-stream), *without* retrieving earlier portions of the stream. The stored media case involves retrieving the *entire* media object. Thus the notion of “startup delay” differs in the two scenarios (i.e., joining an existing stream versus starting a new stream). Third, the peers in a live streaming scenario have a *shared temporal content focus*, while the stored media case has *greater temporal diversity* of requests. The peer dynamics resemble those of file downloading, while still requiring low startup delays for the sequential playback of large media objects. Finally, live streaming implicitly involves sustained content delivery at the intrinsic *media playback rate*, while the stored media case is general: the retrieval rate could vary (e.g., slower than, faster than, or the same as the media playback rate).

These characteristics can challenge the performance of existing P2P protocols. For example, BitTorrent improves the efficiency of file downloads by using a “Rarest-First” piece selection policy to increase the diversity of pieces available in the network. However, streaming protocols require in-order playback of media content, which naturally implies that in-order retrieval of pieces is desirable (but not strictly required). In-order collection of pieces may reduce the spatial and temporal diversity of pieces in a P2P network, resulting in poor system performance.

In this paper, we analytically characterize the performance of BitTorrent-like protocols for on-demand streaming of stored media files. The research questions motivating our work are:

- Can BitTorrent-like protocols provide scalable and efficient on-demand streaming for stored media files?
- How sensitive is performance to the configuration details (e.g., piece selection policies, upload bandwidth) of the P2P application?
- What is the user-perceived performance (e.g., startup delay, playback quality) in such a P2P network?

SIGMETRICS'08, June 2–6, 2008, Annapolis, Maryland, USA.

These questions span network, application, and user-level performance issues.

The main contributions in our paper are the following:

- We show that the analysis of P2P media streaming is decomposable into *download progress* and *sequential progress*, which can be analyzed separately. Furthermore, improving one component can be done without compromising the other.

- We develop detailed analytical models that explicitly consider piece selection policies. The models accurately predict the transition rate of downloaders to seeds, and hence the steady-state swarm population and population mix. The models also provide important insights into the efficiency of a P2P network for on-demand media streaming.

- The models explicitly consider the number of upload and download connections, rather than just the total network bandwidth [21, 24]. This formulation provides the flexibility to model concurrent connections and consider the effects of network asymmetries on the system performance.

- The models provide estimates of the expected retrieval time as well as the variability of the retrieval time for stored media objects. With these models, we can determine suitable tradeoffs between startup delay and the probability of uninterrupted streaming.

The remainder of the paper is organized as follows. Section 2 presents a brief description of the BitTorrent system and a summary of relevant related work. Section 3 explains the derivation of basic models for simple piece selection policies. Section 4 derives detailed models for other piece selection policies, and analyzes the startup delay and retrieval time variability. Section 5 presents simulation results to validate the models. Section 6 summarizes our observations, and Section 7 concludes the paper.

2. BACKGROUND AND RELATED WORK

2.1 BitTorrent

BitTorrent [7] is a popular peer-to-peer file sharing system used to facilitate efficient downloads. BitTorrent splits files into pieces, which can be downloaded in parallel from different peers. BitTorrent distinguishes between peers that have the entire file (called *seeds*), and peers that only have parts of the file (called *leechers* or *downloaders*) and are still downloading the rest of it. The set of peers collaborating to distribute a particular file is known as a BitTorrent *swarm.*

A *tracker* maintains information about the peers participating in a swarm. New peers wanting to download a file are directed (typically using information provided in a metafile available at an ordinary Web server) to a tracker, which provides each new peer with the identity of a random set of participating peers. Each peer typically establishes persistent connections with a large set of peers, consisting of peers identified by the tracker as well as by other peers to which the peer is connected. The peer maintains detailed information about which pieces the other peers have.

While peers typically request pieces from all connected peers that have useful pieces, each peer only uploads to a *limited* number of peers at any given time. Most peers are *choked*, while the peers that it is currently willing to serve are *unchoked.* To encourage peers to upload pieces, BitTorrent uses a rate-based *tit-for-tat* policy, in which downloaders give upload preference to peers that provide high download rates. To probe for better pairings (or in the case of seeds, to allow a new peer to download pieces), each peer periodically unchokes a randomly chosen peer.

To ensure high piece diversity, unchoked peers use a Rarest-First policy to determine which piece to request from the uploading peer [7]. This policy gives strict preference to pieces that are the rarest among the set of pieces owned by all the peers from which a peer is downloading (with ties broken randomly). This approach is very efficient for *file downloading* [15, 16, 21]. In this paper, we consider the efficiency of this and other approaches for *on-demand streaming.*

2.2 Related Work

Prior work on peer-to-peer (or peer-assisted) streaming can be classified into either *live streaming* or *on-demand streaming.* These systems typically use either a *tree-based* or a *data-driven* approach. Tree-based approaches are typically based on application-level multicast architectures, in which the data is propagated through one or more relatively static spanning trees. Such application-level solutions have mainly been used for live streaming [4, 12]. Related tree-based approaches using cache-and-relay [2, 8, 19, 22] have also been proposed for on-demand streaming. In cache-and-relay systems, each peer receives content from one or more parents and stores it in a local cache, from which it can later be forwarded to clients that are at an earlier playback point of the file. The tree-based approaches work best when peer connections are relatively stable.

In the data-driven approach, distribution paths are dynamically determined based on data availability. By splitting the file into smaller parts, each of which may take a completely different path, data-driven protocols can function effectively in dynamic environments (e.g., where peers may join and/or leave the system frequently, and peer connections are heterogeneous, with highly time-varying bandwidths). While most such protocols have been designed for live streaming [17, 25, 26], recently protocols and policies for on-demand streaming have also been proposed [1, 3].

With most peers at similar playback points, peers in live streaming can typically exchange pieces effectively using a relatively small window of pieces. In contrast, with on-demand streaming systems, peers may be at very different playback points. While *download* systems benefit from high piece diversity (as achieved by the Rarest-First policy), in the *streaming* context it is more natural to download pieces in sequential order. To achieve a compromise between these two objectives, Annapureddy *et al.* [1] propose splitting each file into sub-files, with each encoded using distributed network coding [10], and downloaded using a BitTorrent-like approach. By downloading sub-files sequentially, playback can begin after the first sub-file has been retrieved.

Rather than statically splitting each file into sequentially retrieved sub-files, Carlsson and Eager [3] propose a probabilistic piece selection policy with bias to earlier pieces. Using simulations, the authors show that a Zipf-based selection policy achieves a good compromise between high piece diversity and sequential progress. Alternative probabilistic approaches have also been proposed [5]. In this paper, we

Table 1: Model Notation

Parameter	Definition
M	Number of pieces of the target file
U	Maximum number of simultaneous upload connections by a peer
D	Maximum number of simultaneous download connections by a peer
C	Throughput per connection
λ	Arrival rate of new downloading peers into the system
$1/\mu$	Seed residence time in the system
η	System efficiency (file sharing effectiveness)
$x(t)$	Number of downloaders (leechers) in the system at time t
$y(t)$	Number of seeds in the system at time t
T	Download latency for retrieving the complete media file
r	Media playback rate of the file
τ	User-perceived startup delay before playback commences

provide an analytic framework to capture the streaming performance of piece selection policies, including probabilistic approaches.

Many analytic fluid models have been developed to capture the average and transient performance of BitTorrent-like download systems [6, 11, 21]. Assuming that the upload bandwidth is evenly shared among all downloading peers (of a particular class), these models typically consider the steady-state performance, or use differential equations to capture the evolution of the peers (and their performance), given some set of initial conditions or boundary constraints. Other analytic models have captured the interaction of peers at different stages of their download progress [23], characteristics related to the user behavior and peer selection strategies [18, 20], and the minimum time it takes to distribute the file from a server to a set of leechers [14]. Designed for the download context, these models do not capture the order in which pieces are retrieved and therefore cannot be used to compare different piece selection policies.

Closely related to the analysis in this paper are a stochastic fluid model [13] and a probabilistic model [27] used to capture the performance of *live* streaming systems. By capturing the buffer requirements of the average peer, these models can be used to determine how long a newly-arrived client must buffer data before commencing playback. In contrast to the aforementioned, we characterize the system behavior of peer-to-peer on-demand streaming systems. Our models consider both the file sharing effectiveness (which is typically improved by increased piece diversity [15, 18]) and the sequential-order requirements of the streaming media player. Our analysis focuses on the startup delay that can be achieved when using policies in which pieces are retrieved in order. Our models also predict the average download times and the steady-state and transient system population.

3. BASIC SYSTEM MODELS

This section derives simple models to characterize the properties of P2P media streaming in BitTorrent-like networks. Detailed models for different piece selection policies follow in Section 4.

3.1 Model Assumptions

We consider a single swarm (file) in a BitTorrent-like system with seeds and downloaders. Without loss of generality, we assume that this file is of unit size. The parameters and the notation used in our system model are summarized above in Table 1.

The target download file is divided into M pieces, and is encoded for playback at rate r. Each peer is allowed U concurrent upload connections and D concurrent download connections. Each connection achieves mean throughput C. We assume that $D > U$.

We use $x(t)$ to denote the number of downloaders in the system at any time t, and $y(t)$ for the number of seeds [21]. For simplicity of notation, we will use x instead of $x(t)$ and y instead of $y(t)$ when the context is clear. The downloaders download as well as upload data. Seeds (by definition) have all M pieces of the target file, and only upload, with a maximum of U simultaneous uploads. We assume that M is large so that we can ignore the "end game" effects [18] for downloaders with more than $M - D$ pieces.

One objective of our models is to assess the overall effectiveness of different piece selection strategies for media streaming. For mathematical tractability, we introduce a number of simplifying assumptions. We ignore detailed effects of BitTorrent's unchoking and "tit-for-tat" policies. In fact, tit-for-tat is not applicable in systems in which pieces are downloaded strictly in-order (see Section 4.1 for details). We also assume that peers are cooperative; they upload to their best ability, and do not try to cheat the system. They also download the entire file. While we make these simplifying assumptions, we do, however, consider the *system efficiency* η (called *file sharing effectiveness* in [21]). Specifically, η represents the probability that available upload connections are effectively utilized.

New downloaders enter the system at a rate λ, and take time T to complete the download of all M pieces of the file. The latency to begin playback is τ. Note that downloaders become seeds at a rate x/T. Seeds reside in the system for time $1/\mu$, and then leave the system at a rate μy.

We refer to the swarm system as *demand-driven*, since the total demand for download connections exceeds the supply of upload connections; that is, $xD > (x + y)U$. This assumption is appropriate for the default configuration of BitTorrent, and for most network environments, including asymmetric network access technologies such as ADSL.

3.2 Baseline Model: Rarest-First

In this section, we characterize the system behavior when downloaders use the Rarest-First piece selection policy (the default in BitTorrent). The derivation here follows the fluid modeling approach of Qiu and Srikant [21], laying the foundation for our detailed models later in the paper.

A downloader can have D concurrent download connections and U simultaneous upload connections. Each downloader sends requests to seeds and to other downloaders. At any given time, the downloading demand of each downloader is D. However, due to the finite supply of upload connections, a downloader might only acquire n download connections ($0 \leq n \leq D$). We assume that all U upload connections are fully utilized for all peers. In other words, η is 1 in this scenario. In the next section, we show that η is close to 1 for most practical cases.

The downloaders enter the swarm system at a rate λ, and get converted to seeds at a rate $(x + y)UC$. Seeds serve the system for the duration $1/\mu$ and depart at a rate μy. Therefore, the change of population can be expressed as:

$$\frac{dx}{dt} = \lambda - (x + y)UC, \tag{1}$$

$$\frac{dy}{dt} = (x + y)UC - \mu y. \tag{2}$$

Solving the above differential equations for $\frac{dx}{dt} = \frac{dy}{dt} = 0$, we can obtain the average number of downloaders $\overline{x}$ and seeds $\overline{y}$ in steady-state. Specifically, we obtain:

$$\overline{x} = \lambda \left[\frac{1}{UC} - \frac{1}{\mu}\right] \qquad \overline{y} = \frac{\lambda}{\mu} \tag{3}$$

The steady-state results show that:

- The number of downloaders and seeds in the system is *linearly dependent* on the peer arrival rate.

- The number of seeds in the system is *linearly dependent* on the seed residence time ($\frac{1}{\mu}$). As the residence time increases, the number of seeds also increases.

- The total swarm population ($\overline{x} + \overline{y} = \frac{\lambda}{UC}$) is *independent* of the seed residence time; it depends only on the peer arrival rate and the upload capacity of the peers.

The average download latency T can be directly computed using Little's law. Specifically:

$$T = \frac{\overline{x}}{\lambda} = \frac{1}{UC} - \frac{1}{\mu}. \tag{4}$$

This expression shows that the expected download time in steady-state is independent of the peer arrival rate. This result demonstrates why BitTorrent-like systems are inherently scalable [21]. As expected, we find that peers benefit when the upload capacity increases. We also find, as expected, that the download latency decreases as the seed residence time (and thus the number of seeds in the system) increases. However, we do not allow the seed residence time to become arbitrarily large, because our derivation assumes a demand-driven system.

3.3 System Efficiency

This section derives the system efficiency η (also called file sharing effectiveness [21]) for the Rarest-First piece selection policy. The following derivation assumes that each peer knows which pieces are available at other peers in the system, and peers try to download the pieces they need. We are interested in determining what fraction of the available upload connections are used.[1] We will show that for most scenarios η is close to 1.

The Rarest-First piece selection policy attempts to make each file piece equally prevalent amongst the peers. When a peer connects with other peers, it tries to download a needed piece that is rarest among the pool of pieces available from its neighboring peers. The aforementioned strategy asymptotically results in a uniform and identical distribution of file pieces at the peers. Neophyte peers have 0 pieces, while the most senior peers have almost M pieces. Therefore, the probability of finding a particular piece at a (downloader) peer is 1/2.

Consider a single piece of the file. Since the aggregate demand in the system at any time is xD, the average demand for any single piece is $\frac{xD}{M}$. This piece is available from all y seeds and, on average, from only $x/2$ downloaders. Hence, the demand on the potential providers for each piece is:

$$d_s = \frac{2xD}{(x + 2y)M}. \tag{5}$$

Downloaders with only one piece receive demand d_s from other peers. Downloaders with two pieces receive demand $2d_s$, and so on. Thus, the number of idle connections at downloaders with i pieces is $U - id_s$, where $0 \leq i \leq k$ and $k = \lfloor \frac{U}{d_s} \rfloor$.

Due to the uniform distribution of pieces among the downloaders, it follows that the number of downloaders with i pieces is x/M, where $0 \leq i \leq M$. Therefore, the number of idle connections across all downloaders is:

$$n_{idle} = \frac{x}{M}\sum_{i=0}^{k}(U - id_s) \approx \frac{x}{M}\left[kU - \frac{k^2 d_s}{2}\right] = \frac{U^2}{4D}(x + 2y) \tag{6}$$

The system efficiency (file sharing effectiveness) is:

$$\eta = 1 - \frac{n_{idle}}{(x + y)U} = 1 - \frac{U}{4D}\frac{(x + 2y)}{(x + y)} \tag{7}$$

From the demand-driven system assumption, the number of uploads U per peer is less than the number of downloads D per peer. Also, the number of seeds in the system is typically a (small) fraction of the system population. Thus, for most scenarios of interest, $(x + 2y)U \ll (x + y)4D$, which makes η close to 1. (Simulation experiments are consistent with this claim, typically yielding η values of 0.92 or higher.)

3.4 Sequential Progress and Startup Delay

This section introduces the concept of *sequential progress*, which refers to the ability of a piece selection policy to acquire the initial pieces from the beginning of a file, as required for streaming media playback. Note that sequential progress (the sequentiality of the pieces obtained) is independent of the download progress (the rate at which the pieces are obtained).

We consider two simple policies in this section, namely strict *In-Order* retrieval and *Random* piece selection. An example of sequential progress for each is shown in Figure 1.

[1] Our analysis of system efficiency is *connection-centric.* Qiu and Srikant [21] use a *piece-centric* analysis, focusing on the probability of a peer finding a useful piece at another peer, given a connection between these peers. Their analysis ignores the effects of upload/download constraints on system efficiency. Asymptotically, both derivations provide similar results.

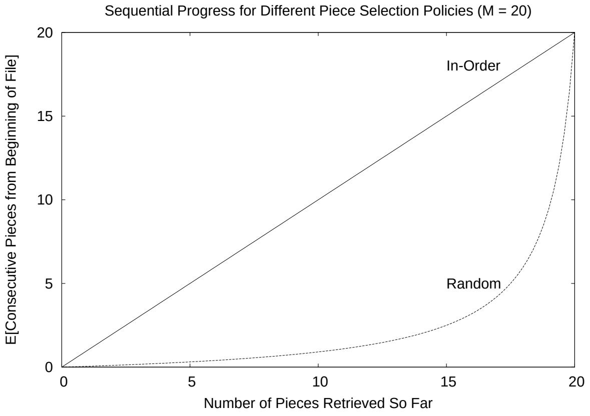


Figure 1: Sequential Progress Example for $M = 20$

By definition, the strict in-order policy is ideal in terms of sequential progress. Each peer simply retrieves the file pieces in numerical order from 1 to M. However, the overall performance of this policy in a P2P network can be sluggish, as will be seen in Section 4.1.

The Random piece selection policy provides poor sequential progress, as shown in Figure 1. While not the worst case[2] for sequential progress, the Random policy provides a useful bound, since no practical piece selection policy would perform worse than Random. Also, we conjecture that Rarest-First performs similarly to Random for sequential progress, since it ignores the numerical ordering of the pieces. (Our simulation results are consistent with this hypothesis.)

The analysis of sequential progress for the Random piece selection policy proceeds as follows. Assume that the file of interest has M pieces, numbered from 1 to M. The downloader retrieves one piece per unit time using a BitTorrent-like protocol, with the pieces chosen uniformly at random.

The question of interest is: "After having downloaded k pieces, what is the probability that a peer has retrieved pieces 1 through j inclusive?" (i.e., j consecutive pieces from the start of the file, useful for streaming). This is called the "sequential progress".

The answer is:

$$Prob(M,k,j) = \frac{\binom{M-j}{k-j}}{\binom{M}{k}} \qquad 1 \le j \le k \le M. \tag{8}$$

The denominator represents the number of distinct ways to choose the k pieces equiprobably at random. The numerator represents the number of ways to choose these pieces such that j are useful pieces (at the start of the file) and $k - j$ are "useless" pieces[3] (not at the start of the file).

The expected value of j can then be computed as:

$$E[j] = \frac{k}{M-k+1}. \tag{9}$$

This is the expression plotted for the Random piece selection policy in Figure 1. Note that after retrieving 1 piece, the probability of having the initial piece of the file is $E[j] = 1/M$, as expected. Similarly, after M pieces are retrieved, the sequential progress from 1 to M is complete ($E[j] = M$).

Further analysis shows that:

$$Var[j] = \frac{k(M+1)(M-k)}{(M-k+2)(M-k+1)^2}. \tag{10}$$

The variance is a monotonically increasing function of k, though it degenerates to 0 when all M pieces are retrieved.

This analysis provides several insights for the Random piece selection policy:

- About *half* of the file ($(M+1)/2$ pieces) must be retrieved before $E[j] \ge 1$. Even after retrieving $M - 1$ pieces, the (expected) sequential progress ($E[j] = (M-1)/2$) is *at most half the file.* This is bad news for on-demand streaming, but makes sense intuitively, since the sole missing piece is equally likely to be in either half of the file, and in the middle on average.

- The sequential progress rate is a *monotonically increasing* function of k. Progress is slow initially, but improves with time as missing holes are filled and large portions of the file become ready for playback.

- *Startup delay can be directly calculated* from the sequential progress. If the media playback rate is r pieces per unit time, then a tangent line of slope r touches the (continuous) sequential progress curve at $k = M + 1 - \sqrt{(M+1)/r}$. The sequential progress rate at this point is $\sqrt{(M+1)r} - 1$ and the absolute startup delay τ is $(M+1)r - 2\sqrt{(M+1)r} + 1$. For example, if $r = 1$, then the startup delay (relative to M) is:

$$1 - \frac{2(\sqrt{(M+1)} - 1)}{M}. \tag{11}$$

- *Startup delay gets worse as M increases.* For $M = 1$, the relative startup delay above is effectively zero. For $M = 20$, the delay would be about 60% (see Figure 1). For $M = 100$, it is 80%. As M approaches infinity, the relative startup delay approaches 1.

Another important observation is the *large gap* between the sequential progress curves for the Random and In-Order policies. There could be many piece selection policies[4] that can provide lower startup delay than Random (or Rarest-First), without requiring strict In-Order retrieval.

4. DETAILED SYSTEM MODELS

This section presents detailed fluid flow models for two variants of the In-Order piece selection policy. We characterize the steady-state system behavior, and discuss startup delay and the variability of download latency.

4.1 Strict In-Order Piece Selection

In this section, we model a BitTorrent-like system for streaming media where downloaders obtain pieces of the media file in strictly sequential order. Downloaders use their

[2] The worst case would be retrieving the pieces in reverse numerical order, from M to 1.

[3] Technically, this expression gives the probability of having *at least* the first j useful pieces, since it is possible that piece $j + 1$ is among the useless pieces (and thus useful). Simple subtraction of the corresponding expressions for j and $j + 1$ gives the probability of having exactly j useful pieces.

[4] For example, the sequential progress for the Zipf piece selection policy recommended in [3] lies approximately halfway between In-Order and Random.

local knowledge to request pieces in (global) numerical order from their connected peers.

The downloading process proceeds in rounds. In each time step, a downloader can issue D concurrent requests (for the next D pieces required). A subset of these requests may be satisfied in a given round.

For this analysis, we make use of peer relationships in the time domain. Each peer arrives at some arbitrary time t and completes download at time $t+T$, on average. We assume that all peers progress through the system in a statistically similar fashion. From the viewpoint of any given peer, there are "younger" peers who arrived later, and there are "older" peers who arrived earlier.

One consequence of strict in-order download is that it *breaks* the "tit-for-tat" reciprocity of BitTorrent (at least within a single swarm). That is, peer relationships are asymmetric, and a downloader never uploads to its provider peer. The asymmetry happens because a given peer can only download from *older* peers (with more pieces of the file), and can only provide content to *younger* peers (with fewer pieces of the file).

Since all downloading requests have the same priority, an uploader that receives more than U requests simply chooses at random U recipients for service. To prevent an infinite backlog from building up in the system, the remaining unsatisfied requests are *purged* from the system, and issued anew in the next round. (Section 4.2 considers the case where the unsatisfied requests are queued.)

For simplicity, we consider the average behavior of the system in steady-state, and disregard details of the end game. Without loss of generality, consider a specific peer that has been in the system for time t_m, where $0 \leq t_m \leq T$. We want to know the probability that such a peer is successful in obtaining a download connection for its next desired piece. That is, we want to compute:

$$p(t_m) = \frac{\tilde{U}(t_m)}{\tilde{D}(t_m)}, \tag{12}$$

where $\tilde{U}(t_m)$ and $\tilde{D}(t_m)$ are the connection supply and demand, respectively, at time t_m.

The peer of age t_m requests download connections from older peers (age $t > t_m$), and from seeds. The total supply of upload connections available for this peer is:

$$\tilde{U}(t_m) = (x + y - \lambda t_m)U. \tag{13}$$

We need to determine the total demand $\tilde{D}(t_m)$ for these connections to evaluate $p(t_m)$. The computation of $\tilde{D}(t_m)$ relies on two observations. First, the total number of download requests in the system is xD. Second, downloading requests are *unevenly* distributed amongst the peers. In particular, peers with more pieces (including seeds) receive higher demand.

$\tilde{D}(t_m)$ can be calculated indirectly by determining the total number of download requests handled by peers *younger* than t_m. Consider a small set of λdt peers at an infinitesimal time interval dt at offset $t < t_m$ (i.e., they have been in the system for t time units). These λdt peers generate $\lambda D dt$ requests, and these requests are spread across $x+y-\lambda t$ peers (downloaders and seeds) in the system. Therefore, the request load on peers younger than t_m (i.e., peers with age in the interval $[t, t_m]$) from the λdt peers is:

$$\frac{\lambda D \ dt}{x + y - \lambda t} \ \ (t_m - t) \ \lambda. \tag{14}$$

The first factor represents the demand per peer, while the second factor represents the number of peers in the region of interest $(t_m - t)$. Therefore, the total request load handled by peers younger than t_m is:

$$\begin{aligned} & \int_0^{t_m} \frac{\lambda^2 \ D \ \ (t_m - t)}{x + y - \lambda t} dt & (15) \\ = \ & \lambda D t_m - (x + y - \lambda t_m) D \ \ln \frac{x+y}{x+y-\lambda t_m}. & (16) \end{aligned}$$

This is the portion of the total demand xD that can be *ignored*, since it does not compete for the supply $\tilde{U}(t_m)$ of upload connections for the reference peer at time t_m. Thus:

$$p(t) = \frac{(x+y-\lambda t)U}{xD - \lambda D t + (x+y-\lambda t) D \ \ln \frac{x+y}{x+y-\lambda t}}. \tag{17}$$

For ease of presentation, we introduce into the numerator a factor $\alpha \geq 1$ to approximate[5] the pro-rated load effect in the denominator of Equation 17. With this notational convenience, the probability that a downloader of age t gets a download connection is:

$$p(t) = \frac{\tilde{U}(t)}{\tilde{D}(t)} = \alpha \frac{(x+y-\lambda t)U}{xD}. \tag{18}$$

When the total download time for M pieces is T, the average downloading rate for a downloader is:

$$\begin{aligned} \gamma &= \frac{1}{T}\int_0^T D p(t) C dt \\ &= \frac{1}{T}\int_0^T \alpha \left(\left(\frac{x+y}{x}\right) UC - \frac{\lambda t UC}{x} \right) dt \\ &= \alpha \left(\frac{x+y}{x}\right) UC - \alpha \frac{\lambda T UC}{2x} \\ &= \alpha UC \left[\left(1 + \frac{y}{x}\right) - \frac{\lambda T}{2x} \right]. \end{aligned}$$

Due to Little's Law, $\overline{x} = \lambda T$. Hence,

$$\gamma \ = \ \alpha UC \left(\frac{y}{x} + \frac{1}{2} \right). \tag{19}$$

Therefore, the change of the number of downloaders and seeds in steady-state can be expressed as follows,

$$\frac{dx}{dt} = \lambda - \gamma x = \lambda - \alpha \left(\tfrac{1}{2}x + y\right) UC, \tag{20}$$

$$\frac{dy}{dt} = \gamma x - \mu y = \alpha \left(\tfrac{1}{2}x + y\right) UC - \mu y. \tag{21}$$

Solving these differential equations for $\frac{dx}{dt} = \frac{dy}{dt} = 0$, we can obtain the steady-state results for downloaders and seeds in the swarm. Specifically, we obtain:

$$\overline{x} = 2\lambda \left[\frac{1}{\alpha UC} - \frac{1}{\mu} \right] \qquad \overline{y} = \frac{\lambda}{\mu} \tag{22}$$

[5] Numerical experiments in Maple show that α is in the range [1.09,1.25] for typical scenarios.

The average download time T can be obtained as follows:

$$T = \frac{\overline{x}}{\lambda} = 2\left[\frac{1}{\alpha UC} - \frac{1}{\mu}\right] \qquad (23)$$

Multiple insights emerge from this analysis:

- This strict In-Order piece selection policy is *sluggish* compared to Rarest-First. This is evident by comparing Equation 20 to Equation 1; one term in the conversion rate differs by approximately $\frac{1}{2}x$.
- The average download latency *almost doubles* compared to Rarest-First (assuming $\alpha \approx 1.1$). This is a large price to pay for the benefit of In-Order retrieval. Similar to the baseline model, latency improves when upload bandwidth and seed residence time are increased.
- The number of downloaders in steady-state *almost doubles* compared to Rarest-First. However, the number of seeds remains the same as in Rarest-First.
- Unlike the Rarest-First policy, the *total swarm population depends on the seed residence time.* In particular, increasing the seed residence time increases the number of seeds in steady-state, while the overall swarm population *decreases.*

Intuitively, the sluggishness of the In-Order policy arises for two reasons. First, the request load is unevenly distributed throughout the network. Older peers with many pieces receive requests from many younger peers, but can only serve U of them; the remaining requests are unfulfilled, and could be re-issued many times before they are served. Young peers with few pieces receive few requests from younger peers; their idle upload connections are wasted, and system efficiency suffers. Second, the purging model allows *all* peers to compete *equally* for service at providers (including seeds), since D requests are issued in each time slot and recipients are chosen *randomly.* As a result, young peers can consume scarce upload connections at seeds and senior peers, impeding the progress of middle-aged peers.

As an aside, the notion of "steady-state" for the strict In-Order policy is debatable. Analysis shows that the number of successful downloads per round by a peer depends on the age of the downloader. Young peers have many providers to choose from, and progress quickly. However, progress becomes slower as peers get older, since there are fewer providers available. (In their simulation results, the authors of [3] note that the swarm population exhibits sawtooth behavior, with highly synchronized conversions of downloaders to seeds. Our analysis helps explain this phenomenon.)

The next section studies a variant of In-Order(Random) that overcomes these problems.

4.2 Strict In-Order Piece Selection (FCFS)

In this section, we consider a variant of the foregoing In-Order piece selection model. Specifically, the model presented here assumes that the uploading peers do *not* purge the unfulfilled requests after each round. Rather, the pending requests at a providing peer are *queued* until they are serviced. The request queues are serviced using First-Come-First-Serve (FCFS).

Unlike the previous model, where peers were serviced randomly, here a peer is guaranteed to obtain service after a finite waiting period (because the request maintains its position in the request queue). Another difference is the intensity with which requests are generated by peers. In the purging model, each downloader issues D requests per round; in the queue-based model, downloaders operate in a closed-loop fashion, with at most D outstanding requests at any time.

If we observe this BitTorrent-like system at any given time, we will find peers with differing degrees of progress. Peers that arrived earlier will have obtained more pieces. The more pieces that a peer has, the more younger peers it can serve. Consequently, we expect to see longer request queues at older peers (and seeds) than at younger peers. Younger peers will provide faster response to requests, although they have fewer pieces available to provide.

Both the finite queue and the FCFS service model[6] are crucial aspects of this system. With these mechanisms, there is conservation of the (finite) request load in the system, as well as bounded delay and fair progress for all downloaders, without any starvation. Furthermore, young peers that indiscriminately send many requests to seeds will experience slow response time for these requests; the closed-loop policy provides a built-in self-regulation mechanism to protect other peers.

In such a P2P network, an emergent phenomenon that we call the "daisy-chain effect" is possible. Imagine a system in which peers of age t download their needed pieces from peers of age $t + \Delta$, and those peers in turn download their needed pieces from peers of age $t + 2\Delta$, and so on. Such a system is still demand-driven, but highly efficient, since all upload connections can potentially be used. This clustering of peers based on their download preferences has been studied in recent work by Gai *et al.* [9], where this phenomenon is called "stratification"; the reader may refer to their paper for detailed analytic characterization of this effect.

Our model does not mandate this structure, but we believe that it is a natural outcome for a self-organizing P2P system. There are at least three ways that such a configuration could arise. One approach (not recommended) is to have peers send duplicate copies of each piece download request to *multiple peers.* When the first successful upload is received, the other outstanding requests can be canceled. Clearly, the response received must have come from an older peer (since it had the piece) with a short queue of requests (since it responded quickly); this is a good candidate for future requests. The second approach is simply to measure the responsiveness of peers that provide pieces, similar to how the existing tit-for-tat mechanism works. Over time, a peer can preferentially send requests to peers that provide fast response, and avoid sending requests to peers that provide slow response (unless there are only a few peers that can provide the required piece). A third approach is to use *finite caches*, allowing a peer to discard a piece of the media file after it has been played locally. This approach provides a temporal bound on useful peer relationships.

Regardless of the actual peer relationships formed, we expect all peers to be busy uploading, and therefore, the system efficiency η is close to 1. Since η is 1, we can assume that peers entering the swarm get converted to seeds at rate $(x + y)UC$, similar to the Rarest-First model. Since seeds serve the system for the duration $1/\mu$ and depart at rate μy,

[6] For example, experiments with a "highest numbered piece" service model also show poor system performance, since the progress of young downloaders is impeded.

the change of population is identical to that for the Rarest-First Model (cf. Equations 1 and 2). Consequently, the steady-state swarm population and the average download latency of this In-Order model match those of the Rarest-First model (cf. Equations 3 and 4).

$$\overline{x} = \lambda \left[\frac{1}{UC} - \frac{1}{\mu}\right] \qquad \overline{y} = \frac{\lambda}{\mu} \qquad T = \frac{1}{UC} - \frac{1}{\mu}. \tag{24}$$

The primary difference with respect to the Rarest-First model is the near-ideal sequential progress achieved during the download. This strict In-Order policy can achieve low startup delay for streaming media playback, without being sluggish.

Finally, we conjecture that the Rarest-First model is near optimal for download progress, since this piece selection policy ensures high utilization of the available upload bandwidth of the swarm. Our foregoing discussion established that In-Order(FCFS) piece selection policy may achieve near optimal system efficiency and achieve download latency identical to that of the Rarest-First policy. The key difference, however, is that In-Order downloads provide ideal sequential progress at the peers. This leads us to conjecture that the In-Order(FCFS) policy is *optimal* for on-demand streaming. (Our simulation results are consistent with this claim, but we do not have a formal proof at this time.)

4.3 Startup Delay Characterization

In this section, we characterize the startup delay: the time since the arrival of a peer into the swarm until it begins playback. Once playback begins, uninterrupted operation is desired. However, many streaming applications can tolerate a small fraction of the pieces arriving too late for playback.

Consider a peer that has been in the system for time t. The expected amount of data downloaded by this peer is $\int_0^t DCp(g)dg$, where $p(g)$ is the probability that a peer successfully obtains a download connection at time $g \leq t$. With startup delay τ and playback rate r, the amount of data that must be available at the peer by time t is $(t - \tau)r$. If a tunable fraction, ε, of the total data is allowed to arrive late, the downloading rate should obey the following inequality:

$$\int_0^t DCp(g)dg \geq (1 - \varepsilon)(t - \tau)r. \tag{25}$$

Let us first consider the case of strict In-Order download with random peer selection. Substituting $p(g)$ from Equation 18 provides:

$$\alpha UC \left[(1 + \frac{y}{x})t - \frac{\lambda}{2x}t^2\right] \geq (1 - \varepsilon)(t - \tau)r. \tag{26}$$

Note that the time to download the first piece of the media file places a lower bound on the achievable startup delay. Therefore, the startup delay τ must satisfy the following inequality for strict In-Order downloading:

$$\tau \geq \max_{\forall t} \left[\frac{1}{MC}, t - \frac{\alpha UC((1 + \frac{y}{x})t - \frac{\lambda}{2x}t^2)}{(1 - \varepsilon)r}\right]. \tag{27}$$

The amount of data downloaded by time t is $\alpha UC((1 + \frac{y}{x})t - \frac{\lambda}{2x}t^2)$, which is a concave monotonically increasing function of t for $0 \leq t \leq T$. Setting $t = T$, we obtain:

$$\tau_{min} = \max\left[\frac{1}{MC}, 2\left(\frac{1}{\alpha UC} - \frac{1}{\mu}\right) - \frac{1}{(1 - \varepsilon)r}\right]. \tag{28}$$

For the In-Order(Random) policy, we have the following observations:

- As in other media streaming systems, *startup delay is determined by the download latency and the playback duration* of the file. For cases where the expected time to download the file exceeds the playback duration of the media, the startup delay equals the maximum of the difference between the aforementioned times, and the time to download the first piece of the media file.

- Startup delay *decreases* with increases in the upload capacity of the peers and the seed residence time. Note that as the expected time to download the file decreases, the startup delay is bounded by the time to download the first piece of the media file.

- Startup delay is *independent* of the peer arrival rate. This demonstrates that on-demand streaming scales well in this type of P2P environment.

We now extend the above analysis to the In-Order(FCFS) policy. In this case, the probability, $p(t)$, that a peer will obtain a download connection at time t is roughly independent of t and can be approximated by $\frac{(x+y)U}{xD}$. Thus, the startup latency τ must satisfy:

$$\tau \geq \max_{\forall t} \left[\frac{1}{MC}, t - \frac{UC(1 + \frac{y}{x})t}{(1 - \varepsilon)r}\right]. \tag{29}$$

Because the amount of data downloaded, $UC(1 + \frac{y}{x})t$, by time t is a linearly increasing function of t, we can simplify the above by substituting $t = T$ to obtain the following:

$$\tau_{min} = \max\left[\frac{1}{MC}, \left(\frac{1}{UC} - \frac{1}{\mu}\right) - \frac{1}{(1 - \varepsilon)r}\right]. \tag{30}$$

From the above analysis for In-Order(FCFS) piece selection, we draw the following conclusions:

- The In-Order(FCFS) policy achieves the *lowest startup delay* among the policies considered, because of its low download latency and excellent sequential progress. As with the In-Order(Random) policy, startup delay is dependent on the expected time to download the file and the media playback duration.

- Similar to other policies, startup delay is independent of the peer arrival rate. Startup delay decreases when upload bandwidth or seed sojourn time are increased.

4.4 Variability of Downloads

In previous sections, we derived the expected download latency based on the steady-state system population for different policies. All formulations are based on expected values, not distributions. However, the number of download connections achieved over time is not fixed; it varies with the system population, efficiency, and seed residence time.

In this section, we analyze the variability of the download latency, since high variability can disrupt media playback. Recall from the previous section that download latency can effectively determine the required startup delay when the rate at which the file can be downloaded is less than (or close to) the media playback rate.

There are two possible approaches to this analysis. One approach is to study the probability of obtaining $N \leq M$ pieces within some time t. The other approach is to study the time required to obtain all M pieces.

We start with the first approach, and determine the probability of completing the file retrieval within the expected download latency T. The following analysis assumes that the download process for an individual peer proceeds in rounds. As in previous sections, the probability of getting a single download connection is $p(t)$, and the total demand by a peer for download connections is D. The number of download connections achieved at time t is a Binomial random variable N_t with parameter $p(t)$ and D. The expected number of connections obtained is:

$$E[N_t] = Dp(t),$$

and the corresponding variance is:

$$Var(N_t) = Dp(t)(1 - p(t)).$$

By summing over many rounds, we can obtain the resulting distribution. That is, the number of downloads in the first round is described by the random variable N_0. Similarly, the downloads during the second round is characterized by N_1 and the third round by N_2 and so on. Assuming independence of the random variables N_i and applying the Central Limit Theorem, the total number of downloads achieved by round (or time) t is a random variable Ω_t with expected value $\sum_{i=0}^{t-1} E[N_i]$ and variance $\sum_{i=0}^{t-1} Var(N_t)$.

Now consider the specific case when $p(t)$ is independent of t. This is the case for both In-Order(FCFS) and Rarest-First. For these policies $p = \frac{(x+y)U}{xD}$, and the mean amount of data downloaded by time t can be computed as follows:

$$E[\Omega_t] = \sum_{i=0}^{t-1} DCp(i) = DCp \sum_{i=0}^{t-1} 1 = DCpt. \qquad (31)$$

Consequently, the variance of Ω_t is,

$$VAR(\Omega_t) = \sum_{i=0}^{t-1} DCp(i)(1 - p(i)) = DCp(1 - p)t. \qquad (32)$$

Note that Ω_t is the summation of multiple discrete random variables. As t increases above 30, $F(\Omega_t)$ is a staircase function approaching a normal distribution. The probability of n downloads at time t can be computed from Ω_t where n ranges from 0 to Dt. At time $t = T$ (where T is the expected download time), we get the mean[7] $E[\Omega_T] = DCpT = 1$ and the variance $VAR(\Omega_T) = 1 - DCp^2T = (1 - p)$.

The following inferences can be observed from the distribution of Ω_t:

- Since Ω_T is discrete normal with mean 1, the probability of downloading the entire file within the expected completion time is 0.50. That is, *half of the downloaders complete within the expected time*, while the other half does not.

- The *variance increases with the number of download connections*. That is, more demand D for the scarce supply U leads to greater variability in the download progress achieved.

- The *variance decreases as the number of upload connections is increased*. Increasing the number of upload connections will tighten the distribution of download latency observed.

- Arrival rate does *not* affect the variability, but higher seed residence time *reduces* the variability slightly.

Using a similar approach, we can derive the mean and variance of number of downloads for the In-Order(Random) policy. Approximating the sum of probabilities with an integral and substituting $p(g)$ from Equation 18 provides:

$$\begin{aligned} E[\Omega_t] &= \sum_{i=0}^{t-1} DCp(i) \approx \int_0^t DCp(g)dg && (33) \\ &\approx \alpha UC \left((1 + \frac{y}{x})t - \frac{\lambda}{2x} t^2 \right) && (34) \end{aligned}$$

$$VAR(\Omega_t) \approx E[\Omega_t] - \frac{\alpha^2 U^2 C}{x^2 D} \left((x+y)^2 t - \lambda(x+y)t^2 + \frac{\lambda^2}{3} t^3 \right) \quad (35)$$

However, the derivation of Equation 33 and 35 holds only when download progresses in very regular fashion and all N_t's are mutually independent (e.g., with many seeds). For the In-Order(Random) policy, the N_t's tend to be mutually dependent, at least in a demand-driven system. With more seeds, this dependency reduces. For Rarest-First and In-Order(FCFS), the N_t's are always mutually independent, regardless of the degree of demand in the system.

We now turn to the second approach, and characterize the total download time for M pieces. Let $\tilde{T}$ denote the random variable for the download time. Since Ω_t denotes the amount of data collected by time t, the probability of completing the entire download at exactly time slot t of duration ϵ[8] can be obtained by summing the probability mass function Ω_t in the region from $1 - \frac{DCp\epsilon}{2}$ to $1 + \frac{DCp\epsilon}{2}$, where $DCp\epsilon$ denotes the expected amount of data collected in each time slot. Therefore, the probability $f(t) = Pr(\tilde{T} = t)$ of completing the download at time t is approximately[9]:

$$\int_{1+\frac{DCp\epsilon}{2}}^{1-\frac{DCp\epsilon}{2}} \frac{1}{\sqrt{2\pi DCp(1-p)t)}} e^{-\frac{(g-DCpt)^2}{2(DCp(1-p)t)}} dg \qquad (36)$$

(Figure 3(c) later in the paper shows a sample plot of this density function for parameter values $M = 200$, $U = 4$, $D = 16$, $\lambda = \frac{1}{2}$, $\frac{1}{\mu} = 8$ and $C = \frac{1}{200}$.)

Several insights can be drawn from this analysis:

- The density function for download latency *resembles a normal distribution*, with a slight skew to the right.

- Approximately half of the distribution lies on each side of the mean $T = \frac{1}{DCp}$.

- The variance term $DCp(1 - p)t$ changes with time, unlike the normal distribution.

Understanding variability is important because it affects the probability of interrupted playback due to late (missing) pieces. It also helps determine online rules for startup delay.

[7] This distribution of Ω_T should be interpreted carefully, since it covers the range $[0, DCT]$, where the total amount of data downloaded could theoretically exceed the file size (assumed to be one). Clearly, a peer never downloads more than the file size, but this formulation is still mathematically correct. The cumulative probability in the range $[1, DCT]$ denotes the probability of download completion.

[8] We note that $\epsilon = \frac{1}{MC}$ corresponds to the case when an active connection can upload exactly one piece per round.

[9] We assume a large value of M when using this continuous density function approach.

5. MODEL VALIDATION

In this section, we present ns-2 fluid simulation experiments to validate the analytical models developed in Sections 3 and 4. In these experiments, we assume a homogeneous swarm, in which all peers have identical configuration parameters. Peers arrive to the system continuously, perform a complete download, and remain for a short duration before leaving the system. The peer inter-arrival times are exponential, while the seed residence times are drawn from a normal distribution. The default peer arrival rate is 50 per media playback duration.

The parameter settings in the simulation experiments are as follows. The media file has $M = 100$ pieces, each 128 KB in size. The media playback rate is 2000 Kbps. The peer upload bandwidth ranges from 600 Kbps to 2000 Kbps, while the number of upload connections U ranges from 3 to 15, with a default of 4. Unless stated otherwise, the download bandwidth is 3200 Kbps for $D = 16$ connections (i.e., each connection gets 200 Kbps). The default seed residence time is 20 seconds. All simulation results are normalized to the media playback duration in the default configuration.

Figure 2 presents the results from our simulation experiments. The top row of graphs shows swarm population, while the other two rows show results for download latency and startup delay, respectively.

The simulation results for swarm population show good agreement with the analytical models. Figure 2(a) shows that the total swarm population is linearly dependent on the peer arrival rate, as expected. The three analytical models are presented using lines, as labeled in the graph key. The corresponding simulation results appear as points on the graph, with '+' for Rarest-First, circles for In-Order(Random), and squares for In-Order(FCFS). In-Order(FCFS) behaves similarly to Rarest-First, while In-Order(Random) is sluggish: its swarm population increases at twice the rate of the others. Figure 2(b) shows the swarm population versus the seed residence time. The In-Order(Random) policy has a higher swarm population, but the swarm population decreases (as predicted) when the seed residence time increases. For large seed residence times, the swarm population increases for all three models as seeds become plentiful (i.e., the system is no longer demand-driven). Figure 2(c) shows that the swarm population decreases as the upload bandwidth is increased. Beyond a certain upload bandwidth, the population remains constant, since the download bandwidth becomes the system bottleneck.

The second row of graphs in Figure 2 shows the results for download latency. The analytical models predict that the download time is independent of the peer arrival rate. The simulation results in Figure 2(d) show a similar trend, though the In-Order(Random) policy deviates somewhat from the model prediction. Figure 2(e) considers the effect of seed residence time. For all three models, more seeds in the system means faster downloads. The effect of upload bandwidth is illustrated in Figure 2(f). As expected, increasing the upload bandwidth reduces the download time, until the download bandwidth becomes the bottleneck.

The third row of graphs in Figure 2 shows the startup delay for media playback. The analytical models predict that the startup delay is independent of the peer arrival rate. The simulation results in Figure 2(g) confirm this. Also, the startup delay of In-Order(FCFS) is lower than that of Rarest-First, while In-Order(Random) is much worse. The impact of seed residence time is shown in Figure 2(h). In general, increasing the seed residence time reduces the startup delay. In-Order(FCFS) has the lowest startup delays among the policies evaluated. For both In-Order policies, the startup delay is lower bounded by the piece retrieval time, once the seed residence time is large enough. Rarest-First never reaches this point, because of its poor sequential progress, and the download bandwidth bottleneck in this scenario. Figure 2(i) shows similar trends for the effect of upload bandwidth. In general, increasing the upload bandwidth reduces the startup delay. For both In-Order policies, the startup delay equals the piece retrieval time once the upload bandwidth is high enough.

Figure 3 shows validation results for the variability analysis. In these experiments, we use $M = 200$ pieces. Figure 3(a) shows the distribution for the number of pieces downloaded within the expected download time T, for the Rarest-First policy. The simulation results match the analytical model well. Figure 3(b) shows the simulation results for the distribution of download time. The shape of the distribution matches well with the analytical distribution in Figure 3(c), though the simulation results are about 15-20% higher. We attribute the difference to "end game" effects in the ns-2 fluid simulation model.

6. DISCUSSION

Our analytic and simulation results show that the in-Order(FCFS) policy is well-suited for use in BitTorrent-like on-demand media streaming systems. At the same time, our results also show that this policy achieves the same download latency as the Rarest-First policy.

Our analysis was made possible by the fundamental insight that *media streaming progress* (i.e., the rate at which useful pieces are obtained by a peer for media playback) is essentially the product of *download progress* (i.e., the rate at which pieces are successfully obtained from the P2P network) and *sequential progress* (i.e., the usefulness of the obtained pieces for media playback).

Our characterization of download latency captures the download progress concept. Our analytical models and simulation results show that download progress is primarily influenced by peer population, upload bandwidth, download bandwidth, and piece selection policy.

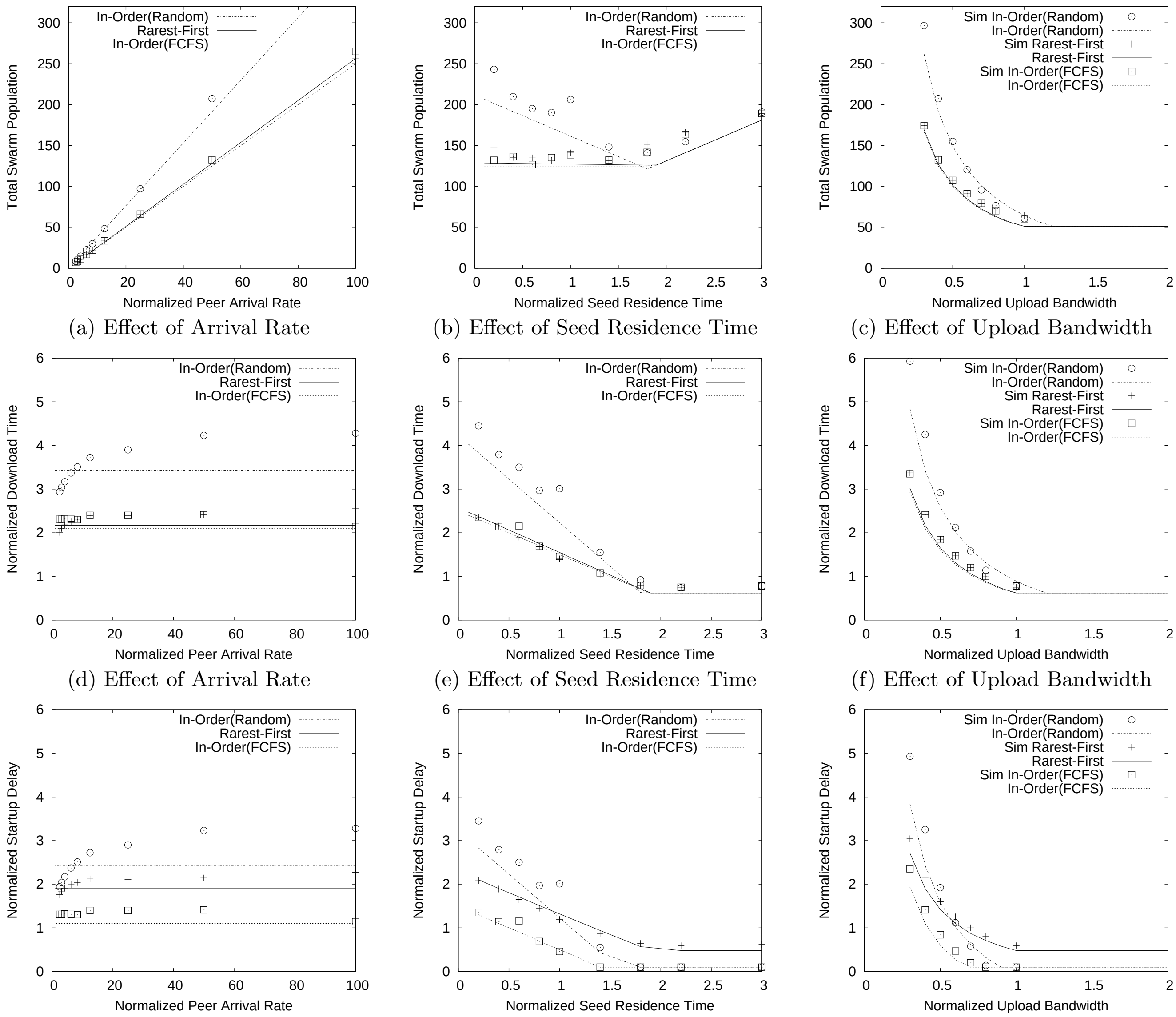
The sequential progress concept is captured by our startup delay characterization. We find that sequential progress is primarily influenced by piece selection policy and the number of pieces in the media.

While both download progress and sequential progress are influenced to some extent by the piece selection policy, and thus are not independent, we believe that these concepts are nonetheless separable. Specifically, we show that sequential progress can be optimized without compromising the download progress of the media file.

7. CONCLUSIONS

In this paper, we developed detailed analytical models to characterize the behavior of BitTorrent-like protocols for on-demand stored media streaming. Our models explicitly capture the effects of different piece selection policies, including Rarest-First and two variants of In-Order.

Our models provide insight into the transient and steady-state behavior of a P2P network used for on-demand stream-

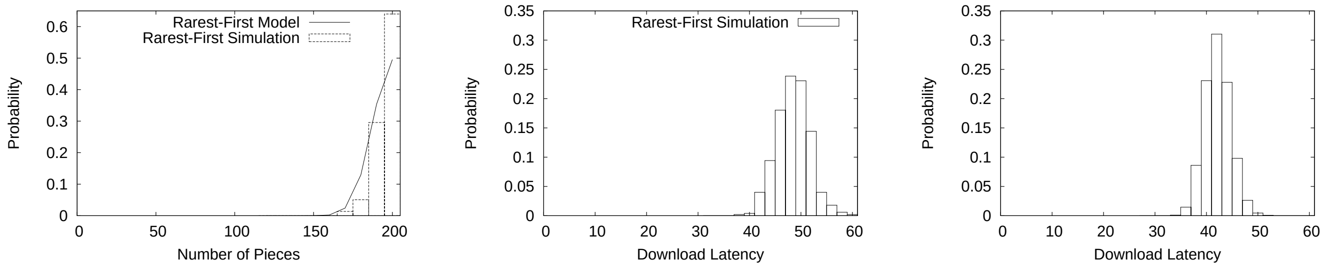


(a) Effect of Arrival Rate (b) Effect of Seed Residence Time (c) Effect of Upload Bandwidth

(d) Effect of Arrival Rate (e) Effect of Seed Residence Time (f) Effect of Upload Bandwidth

(g) Effect of Arrival Rate (h) Effect of Seed Residence Time (i) Effect of Upload Bandwidth

Figure 2: Model Validation Results (analytic results use lines; simulation results use points, with '+' for Rarest-First; circles for In-Order(Random); and squares for In-Order(FCFS)). Top row: Swarm Population. Middle row: Download Latency. Bottom row: Startup Delay.



(a) Pieces Downloaded (b) Download Latency (simulation) (c) Download Latency (analytical)

Figure 3: Validation of Variability Results for Rarest-First

ing. We demonstrate the poor sequential progress characteristics of Random (and Rarest-First) piece selection policies, and motivate the need for In-Order piece selection. We use our model to explain the sluggishness of naive In-Order streaming. In particular, we identify the reduced system efficiency in the purging model with random peer selection, and use these insights to explore the In-Order(FCFS) policy. The latter policy provides the same download latency as Rarest-First, with substantially lower startup delay for media streaming. We conjecture that this policy is optimal for startup delay.

Simulation results are used to validate the models. We compare different retrieval policies across a wide range of system parameters, including peer arrival rate, seed residence time, and upload/download bandwidth. We also provide quantitative results on the startup delays and retrieval times for streaming media delivery. The simulation results show close agreement with the analytical models.

In summary, our results provide valuable insights into on-demand media streaming on P2P networks. Ongoing work is focusing on proofs of the optimality results.

Acknowledgements

The authors are grateful to the anonymous reviewers and our assigned shepherd Dan Rubenstein for their constructive suggestions, which helped improve the clarity of the paper. Martin Arlitt, Phillipa Gill, Zongpeng Li, and Aniket Mahanti provided useful feedback on earlier versions of this paper. Discussions with Derek Eager have also been beneficial to this work. Financial support for this research was provided by NSERC (Natural Sciences and Engineering Research Council) in Canada and iCORE (Informatics Circle of Research Excellence) in the Province of Alberta.

8. REFERENCES

[1] S. Annapureddy, C. Gkantsidis, and P. Rodriguez. Providing Video-on-Demand using Peer-to-Peer Networks. In *Proc. Workshop on Internet Protocol TV (IPTV) '06*, Edinburgh, Scotland, May 2006.

[2] A. Bestavros and S. Jin. OSMOSIS: Scalable Delivery of Real-time Streaming Media in Adhoc Overlay Networks. In *Proc. ICDCS Workshops '03*, pages 214–219, Providence, RI, May 2003.

[3] N. Carlsson and D. L. Eager. Peer-assisted On-demand Streaming of Stored Media using BitTorrent-like Protocols. In *Proc. IFIP/TC6 Networking '07*, pages 570–581, Atlanta, GA, May 2007.

[4] M. Castro, P. Druschel, A. Rowstron, A.-M. Kermarrec, A. Singh, and A. Nandi. SplitStream: High-Bandwidth Multicast in Cooperative Environments. In *Proc. ACM SOSP '03*, pages 298–313, Bolton Landing, NY, October 2003.

[5] Y. R. Choe, D. L. Schuff, J. M. Dyaberi, and V. S. Pai. Improving VoD Server Efficiency with BitTorrent. In *Proc. ACM MULTIMEDIA '07*, pages 117–126, Augsburg, Germany, September 2007.

[6] F. Clevenot-Perronnin, P. Nain, and K. Ross. Multiclass P2P Networks: Static Resource Allocation for Service Differentiation and Bandwidth Diversity. In *Proc. IFIP Performance*, pages 32–49, Juan-les-Pins, France, October 2005.

[7] B. Cohen. Incentives Build Robustness in BitTorrent. In *Proc. Workshop on Economics of Peer-to-Peer Systems '03*, Berkeley, CA, June 2003.

[8] Y. Cui, B. Li, and K. Nahrstedt. ostream: Asynchronous streaming multicast in application-layer overlay networks. *IEEE Journal on Selected Areas in Communications (Special Issue on Recent Advances in Service Overlays)*, 22(1):91–106, January 2004.

[9] A.-T. Gai, F. Mathieu, F. de Montgolfier, and J. Reynier. Stratification in P2P Networks: Application to BitTorrent. In *Proc. ICDCS '07*, Toronto, Canada, June 2007.

[10] C. Gkantsidis and P. R. Rodriguez. Network Coding for Large Scale Content Distribution. In *Proc. IEEE INFOCOM '05*, pages 2235–2245, Miami, FL, March 2005.

[11] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang. Measurement, Analysis, and Modeling of BitTorrent-like Systems. In *Proc. ACM Internet Measurement Conference (IMC) '05*, pages 35–48, Berkeley, CA, October 2005.

[12] D. Kozic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High Bandwidth Data Dissemination using an Overlay Mesh. In *Proc. ACM SOSP '03*, pages 282–297, Bolton Landing, NY, October 2003.

[13] R. Kumar, Y. Liu, and K. Ross. Stochastic Fluid Theory for P2P Streaming Systems. In *Proc. IEEE INFOCOM '07*, pages 919–927, Anchorage, AK, May 2007.

[14] R. Kumar and K. Ross. Peer Assisted File Distribution: The Minimum Distribution Time. In *Proc. IEEE Workshop on Hot Topics in Web Systems and Technologies '06*, Boston, MA, November 2006.

[15] A. Legout, N. Liogkas, E. Kohler, and L. Zhang. Clustering and Sharing Incentives in BitTorrent Systems. In *Proc. ACM SIGMETRICS '07*, pages 301–312, San Diego, CA, June 2007.

[16] A. Legout, G. Urvoy-Keller, and P. Michiardi. Rarest First and Choke Algorithms Are Enough. In *Proc. ACM Internet Measurement Conference (IMC) '06*, pages 203–216, Rio de Janeiro, Brazil, October 2006.

[17] X. Liao, H. Jin, Y. Liu, L. M. Ni, and D. Deng. AnySee: Peer-to-Peer Live Streaming. In *Proc. IEEE INFOCOM '06*, Barcelona, Spain, April 2006.

[18] M. Lin, B. Fan, D. M. Chiu, and J. C. S. Lui. Stochastic Analysis of File Swarming Systems. In *Proc. IFIP Performance '07*, pages 856–875, Cologne, Germany, October 2007.

[19] J.-G. Luo, Y. Tang, and S.-Q. Yang. Chasing: An Efficient Streaming Mechanism for Scalable and Resilient Video-on-Demand Service over Peer-to-Peer Networks. In *Proc. IFIP Networking '06*, pages 642–653, Coimbra, Portugal, May 2006.

[20] L. Massoulie and M. Vojnovic. Coupon Replication Systems. In *Proc. ACM SIGMETRICS '05*, pages 2–13, Banff, Canada, June 2005.

[21] D. Qiu and R. Srikant. Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks. In *Proc. ACM SIGCOMM '04*, pages 367–378, Portland, OR, August 2004.

[22] A. Sharma, A. Bestavros, and I. Matta. dPAM: A Distributed Prefetching Protocol for Scalable Asynchronous Multicast in P2P Systems. In *Proc. IEEE INFOCOM '05*, pages 1139–1150, Miami, FL, March 2005.

[23] Y. Tia, D. Wu, and K. W. Ng. Modeling, Analysis and Improvement for BitTorrent-Like File Sharing Networks. In *Proc. IEEE INFOCOM '06*, Barcelona, Spain, April 2006.

[24] X. Yang and G. Veciana. Service Capacity of Peer to Peer Networks. In *Proc. IEEE INFOCOM '04*, pages 2242–2252, Hong Kong, China, March 2004.

[25] M. Zhang, L. Zhao, Y. Tang, J.-G. Luo, and S.-Q. Yang. Large-scale Live Media Streaming over Peer-to-Peer Networks through Global Internet. In *Proc. Workshop on Advances in Peer-to-Peer Multimedia Streaming '05*, pages 21–28, Singapore, November 2005.

[26] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum. CoolStreaming/DONet: A Datadriven Overlay Network for Peer-to-Peer Live Media Streaming. In *Proc. IEEE INFOCOM '05*, pages 2102–2111, Miami, FL, March 2005.

[27] Y. Zhou, D. Chiu, and J. C. S. Lui. A Simple Model for Analyzing P2P Streaming Protocols. In *Proc. ICNP '07*, pages 226–235, Beijing, China, October 2007.