Insights on Media Streaming Progress using BitTorrent-like Protocols for On-Demand Streaming

Nadim Parvez, Carey Williamson, Anirban Mahanti, and Niklas Carlsson

***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, two variants of In-Order, and two probabilistic policies (Portion and Zipf). Our models provide insight into 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 design tradeoffs for on-demand media streaming in peer-to-peer networks. Finally, the models are validated using both fluid-based and packet-level simulations.**

***Index Terms*—Peer-to-peer systems, BitTorrent, On-demand streaming, Modeling**

I. 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 file sharing networks were originally created.

The P2P paradigm has been used successfully for *live* media streaming, but the (more difficult) case of on-demand streaming of *stored* media has received relatively less 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 stored media case involves retrieving the *entire* media object, while the live streaming case allows peers to join at any time (i.e., mid-stream), *without* retrieving earlier portions of the stream. Thus the issue 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 be slower than, faster than, or the same as the media playback rate, or even vary with time.

C. Williamson is with the Department of Computer Science, University of Calgary, Canada. Email: carey@cpsc.ucalgary.ca A. Mahanti is with National ICT Australia (NICTA), Eveleigh, NSW, Australia. Email:anirban.mahanti@nicta.com.au N. Carlsson is with Linköping University, Linköping, Sweden. Email:niklas.carlsson@liu.se

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. Our models capture performance differences between various policies and configuration details (e.g., piece selection policies, upload bandwidth) and allow us to answer questions related to the efficency and user-perceived performance of BitTorrent-like on-demand streaming protocols.

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 usually 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, as well as the steady-state swarm population size and mix. The models provide important insights into the efficiency of on-demand media streaming in P2P networks.
- The models explicitly consider the number of upload and download connections, rather than just the total network bandwidth [26], [29]. This formulation provides the flexibility to model concurrent connections and consider the effect of network asymmetry on the system performance.
- The models provide estimates of the expected retrieval time for stored media objects, as well as its variability, so that we can determine suitable tradeoffs between startup delay and the likelihood of uninterrupted streaming.
- The models are validated using both fluid-based and packet-level simulations.

The remainder of the paper is organized as follows. Section II presents a brief description of the BitTorrent system, as well as the concepts of download progress and sequential progress. Sections III and IV explain the derivation of basic models for sequential progress and download progress, respectively, using different piece selection policies. Section V presents the analysis of startup delay, and also discusses several extensions of our model. Section VI presents simulation results to validate the models. Section VII summarizes relevant related work, while Section VIII concludes the paper.

II. Background and System Tradeoff

A. BitTorrent

BitTorrent [9] 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 meta-file 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 can 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. That is, most peers are *choked*, while a few 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 [9]. This policy gives preference to pieces that are the least prevalent within a neighbourhood of cooperating peers (with ties broken randomly). This approach is very efficient for *file downloading* [17], [18], [26]. In this paper, we consider the efficiency of this and other piece selection strategies for *on-demand streaming*.

B. Basic System Tradeoff

As discussed in the introduction, there is a fundamental tradeoff between maintaining high piece diversity versus the in-order requirements of streaming. In general, the Rarest-First policy achieves the highest piece diversity, while In-Order policies are the most natural because of the inherent sequential playback requirements associated with video streams.

This section introduces the concept of *media streaming progress (MSP)*, which is defined as the number of useful media pieces obtained per unit time. Conceptually, the MSP can be separated into two parts: (1) the *download progress (DP)*, which is defined as the number of pieces retrieved per unit time, and (2) the *sequential progress (SP)*, which is defined as the number of *useful* in-order media pieces obtained per piece retrieved. The MSP is simply the product of these two metrics. Equation 1 expresses this simple relationship.

$$MSP = DP \times SP \qquad (1)$$

The download progress captures the generic notion of throughput (i.e., a policy's ability to download pieces quickly), while the sequential progress refers to an application-specific property, namely 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 conceptually independent of the download progress (the rate at which the pieces are obtained). In the following sections, we analyze these metrics separately, starting with sequential progress.

III. Sequential Progress

In this section we analyze the *sequential progress* of four simple policies: strict *In-Order* retrieval, *Random* piece selection, and two probabilistic piece selection policies (*Portion* and *Zipf* [3]). An example of sequential progress for each is shown in Figure 1.

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 download progress of this policy in a P2P network can be sluggish, as will be seen in Section IV-D.

The Random piece selection policy provides poor sequential progress, as shown in Figure 1. While not the worst case[1] 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.)

Finally, the two probabilistic piece selection policies (Portion and Zipf) are statistically biased towards earlier pieces. The Portion policy has a single control parameter p. At each step, it chooses pieces according to the In-Order policy with probability p, and according to the Rarest-First policy with probability $1 - p$. With the Zipf policy, a Zipf distribution is used to skew the bias towards selecting earlier pieces. These two policies are carefully defined and discussed in [3].

A. Random Piece Selection

The analysis of sequential progress for the Random piece selection policy proceeds as follows. Assume that the file of

[1]The worst case is retrieving the pieces in reverse numerical order, from M to 1, since playback cannot commence until all M pieces are retrieved.

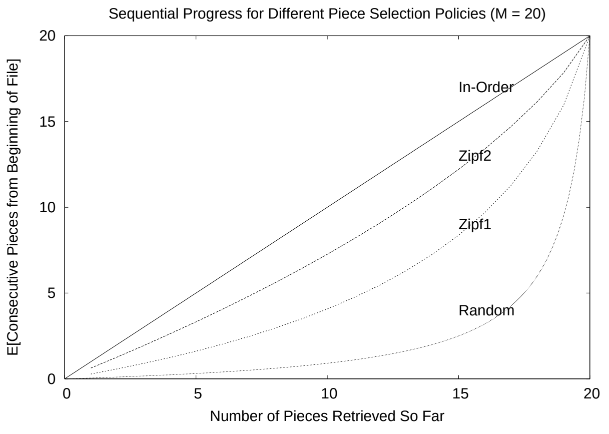


Fig. 1. Sequential Progress Example for $M = 20$

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:

$$P(j,k) = \frac{\binom{M-j}{k-j}}{\binom{M}{k}} \qquad 1 \leq j \leq k \leq M. \tag{2}$$

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[2] (not at the start of the file).

The expected value of j (given k pieces) is:

$$E[j|k] = \frac{k}{M-k+1}. \tag{3}$$

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 $E[j|1] = 1/M$, as expected. Similarly, after M pieces are retrieved, the sequential progress from 1 to M is complete ($E[j|M] = M$).

This analysis leads to several key insights for the Random piece selection policy:

- About *half* of the file ($(M+1)/2$ pieces) must be retrieved before $E[j|k] \geq 1$. This result has major implications on the expected startup delay.
- Even after retrieving $M-1$ pieces, the (expected) sequential progress ($E[j|M-1] = (M-1)/2$) is *at most half the file*. This is bad news for on-demand streaming, since it again has implications on startup delay. However, the result 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.

[2]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.

- 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. This is an encouraging result: once the MSP reaches the threshold required for playback, streaming can certainly commence, and should be able to complete in an uninterrupted fashion.
- *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 relative startup delay is:

$$1 - \frac{2(\sqrt{(M+1)} - 1)}{M}. \tag{4}$$

- *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. That is, streaming degenerates to the download case if the Random policy is used on a large media file with many pieces.

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

B. Portion Piece Selection

We next consider probabilistic policies with bias towards earlier pieces, such as Portion and Zipf [3]. As in the case of the Random piece selection policy, our analysis considers the sequential progress achieved after k out of M pieces have been retrieved ($1 \leq k \leq M$). More specifically, we are interested in the expected sequential progress $E[j|k]$, where j is the number of useful pieces (at the start of the file). For simplicity, we say that a peer is in state (j,k) whenever it has k pieces in total, and the piece with index $(j+1)$ is its first missing piece (i.e., it has the first j in-order pieces, but is missing piece $j+1$).

Let $P(j,k)$ be the probability that we have obtained j in-order pieces when the peer has retrieved k pieces in total. Clearly, at any point in time, the $P(j,k)$ values must satisfy $\sum_{j=0}^{k} P(j,k) = 1$, for a given k. The expected number of in-order pieces $E[j|k]$ can be calculated as

$$E[j|k] = \sum_{j'=0}^{k} j' P(j',k). \tag{5}$$

For simplicity, we (conservatively) assume that all pieces are equally likely to be in the possession of the uploading peer (as is the case with seeders, or when using the Random policy, for example). In a real system, leechers downloading from other leechers are more likely to have pieces that are closer to its playback point, causing a somewhat stronger bias towards earlier pieces.

Furthermore, we assume that all pieces that already have been retrieved, and are not among the first j in-order pieces, are uniformly distributed. With this assumption, a peer in state (j,k) has each of the pieces in the range $[1,j]$, does not have piece $j+1$, and has each of the pieces in the range $[j+2,M]$ with a probability $\frac{k-j}{M-j-1}$. (With the Zipf policy, discussed in the next section, this assumption provides a crude approximation.)

Let $q_{j,k}$ be the probability that the first missing piece is selected for retrieval next. With the Random policy, $q_{j,k} = \frac{1}{M-k}$. With the Portion policy, using a parameter p,

$$q_{j,k} = p + (1-p)\frac{1}{M-k}. \tag{6}$$

This expression uses the assumption that the missing pieces (excluding the first missing piece) are uniformly distributed, and each other piece is equally likely to be missing (and/or selected).

Let $A_{k,j,n}$ be the probability that we have exactly n consecutive in-order pieces starting at position $j+2$ at the time we also have j in-order pieces (starting at piece 1), piece $j+1$ is missing, and we have a total of k pieces. With the above assumption of $k-j$ uniformly distributed pieces in the range $[j+2,M]$, we have

$$A_{j,k,n} = \begin{cases} \left(1-\frac{k-j-n}{M-j-n-1}\right)\prod_{i=0}^{n-1}\frac{k-j-i}{M-j-i-1}, & \text{if } j+n<k \\ \prod_{i=0}^{n-1}\frac{k-j-i}{M-j-i-1}, & \text{if } j+n=k \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

Note that the series of factors in these expressions correspond to conditional probabilities. Specifically, each factor gives the probability that piece $j+2+i$ is (or is not) obtained, conditioned on all pieces in the range $[j+2, j+1+i]$ having been retrieved, in addition to the initial j in-order pieces.

Given the above notation, the probability $P(j,k)$ can now be expressed as follows:

$$P(j,k) = P(j,k-1)(1-q_{j,k-1}) + \\ + \sum_{j'=0}^{j-1}\left[P(j',k-1)q_{j',k-1}A_{j',k-1,(j-j'-1)}\right]. \tag{8}$$

We have validated that the model gives the expected results for two special cases. When $p \to 0$, the results match the model for Random in Section III-A. When $p \to 1$, the results match those for In-Order.

C. Zipf Piece Selection

We now modify the above analysis to handle the Zipf policy [3]. For simplicity, we again assume that the missing pieces (except the first missing piece) are uniformly distributed. Under this assumption, it can be shown that the corresponding expression for $q_{j,k}$ (Equation (6) for the Portion policy) can be approximated as:

$$q_{j,k} \approx \frac{1}{1+\frac{M-k-1}{M-j-1}\sum_{i=2}^{M-j}\frac{1}{i^{\theta}}}. \tag{9}$$

Here, the equal weights of the latter pieces are most noticable in the factor $\frac{M-k-1}{M-j-1}$. The rest of the analysis remains the same as for the Portion policy.

TABLE I
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

As with the Portion policy, we have validated that the model gives the expected results for two special cases. The above model matches the model for Random (Section III-A) when $\theta \to 0$, and matches the In-Order policy when $\theta \to \infty$.

IV. Download Progress

This section derives simple models to characterize the Download Progress properties of P2P on-demand media streaming in BitTorrent-like networks.

A. 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 notation used in our model are summarized in Table I.

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 [26]. 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 [20] for downloaders with more than $M-D$ pieces.

One objective of our model is to assess the overall effectiveness of different piece selection strategies for on-demand 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 retrieved strictly in-order (see Section IV-D for details). We also assume that peers download the entire file, and that they are cooperative; that is, they upload to the best of their ability, and they do not try to cheat the system. While we make these simplifying assumptions, we do, however, consider the *system efficiency* η (called *file sharing effectiveness* in [26]). 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 most network environments, including asymmetric network access technologies such as ADSL.

B. 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 [26], 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 ($0 \le n \le D$). 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:

$$\frac{dx}{dt} = \lambda - (x+y)UC, \tag{10}$$

$$\frac{dy}{dt} = (x+y)UC - \mu y. \tag{11}$$

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

$$\overline{x} = \lambda \left[\frac{1}{UC} - \frac{1}{\mu}\right] \qquad \overline{y} = \frac{\lambda}{\mu} \tag{12}$$

The steady-state results provide the following insights:

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

$$T = \frac{\overline{x}}{\lambda} = \frac{1}{UC} - \frac{1}{\mu}. \tag{13}$$

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 [26]. 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.

C. System Efficiency

This section derives the system efficiency η (also called file sharing effectiveness [26]) 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.[3] 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 (randomly chosen) 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:

$$d_s = \frac{2xD}{(x+2y)M}. \tag{14}$$

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 $0 \le i \le k$ 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 $0 \le i \le M$. Therefore, the number of idle connections across all downloaders is:

$$n_{idle} = \frac{x}{M}\sum_{i=0}^{k}(U - id_s) \approx \frac{x}{M}\left[kU - \frac{k^2 d_s}{2}\right] = \frac{U^2}{4D}(x+2y) \tag{15}$$

The system efficiency (file sharing effectiveness) is:

$$\eta = 1 - \frac{n_{idle}}{(x+y)U} = 1 - \frac{U}{4D}\frac{(x+2y)}{(x+y)} \tag{16}$$

[3]Our analysis of system efficiency is *connection-centric*. That is, given that we know which piece we want from a given peer, we analyze the probability that a connection is available for downloading this piece. Qiu and Srikant [26] use a *piece-centric* analysis, focusing on the probability of a peer finding a useful piece at another peer, given an existing connection between these peers. Their analysis ignores the effects of upload/download constraints on system efficiency. Asymptotically, both derivations provide similar results.

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.)

The following two sections present 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.

D. Naive 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 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 "older" peers who arrived earlier, and "younger" peers who arrived later.

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 (who have more pieces of the file), and can only provide content to *younger* peers (who have fewer pieces of the file).

This simple In-Order model operates in a naive open-loop fashion, and can generate significant load imbalance in the system. 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 IV-E 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 $0 \le t_m \le T$. 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:

$$p(t_m) = \frac{\tilde{U}(t_m)}{\tilde{D}(t_m)}, \tag{17}$$

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:

$$\tilde{U}(t_m) = (x + y - \lambda t_m)U. \tag{18}$$

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:

$$\frac{\lambda D\ dt}{x+y-\lambda t}\ \ (t_m - t)\ \lambda. \tag{19}$$

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:

$$\begin{aligned} & \int_0^{t_m} \frac{\lambda^2\ D\ \ (t_m - t)}{x+y-\lambda t} dt & (20) \\ = \ & \lambda D t_m - (x+y-\lambda t_m) D \ln \frac{x+y}{x+y-\lambda t_m}. & (21) \end{aligned}$$

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:

$$p(t) = \frac{(x+y-\lambda t)U}{xD - \lambda D t + (x+y-\lambda t) D \ln \frac{x+y}{x+y-\lambda t}}. \tag{22}$$

For ease of presentation, we introduce into the numerator a factor $\alpha \ge 1$ to approximate[4] the pro-rated load effect in the denominator of Equation 22. With this notational convenience, the probability that a downloader of age t gets a download connection is:

$$p(t) = \frac{\tilde{U}(t)}{\tilde{D}(t)} = \alpha \frac{(x+y-\lambda t)U}{xD}. \tag{23}$$

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

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

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

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

$$\gamma \quad = \quad \alpha UC \left(\frac{y}{x} + \frac{1}{2} \right). \tag{24}$$

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

$$\frac{dx}{dt} \quad = \lambda - \gamma x = \lambda - \alpha \left(\tfrac{1}{2}x + y\right) UC, \tag{25}$$

$$\frac{dy}{dt} \quad = \gamma x - \mu y = \alpha \left(\tfrac{1}{2}x + y\right) UC - \mu y. \tag{26}$$

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

$$\overline{x} = 2\lambda \left[\frac{1}{\alpha UC} - \frac{1}{\mu} \right] \qquad \overline{y} = \frac{\lambda}{\mu} \tag{27}$$

The average download time T can be obtained as follows:

$$T = \frac{\overline{x}}{\lambda} = 2 \left[\frac{1}{\alpha UC} - \frac{1}{\mu} \right] \tag{28}$$

Multiple insights emerge from this analysis:

- This naive In-Order piece selection policy is *sluggish* compared to Rarest-First. This is evident by comparing Equation 25 to Equation 10; 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, but *decreases* the overall swarm population.

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 reissued many times before they are served. Conversely, young peers with few pieces receive few requests from even 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 the In-Order policy that overcomes these problems.

E. In-Order Piece Selection (FCFS)

In this section, we consider a closed-loop version 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[5] are crucial aspects of this system. These mechanisms conserve the (finite) request load in the system, ensuring 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 by Gai *et al.* [11], 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. Regardless of the actual peer relationships formed, we expect

[5]For example, a "highest numbered piece" service model also exhibits poor system performance, since it impedes the progress of young downloaders.

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, the change of population is identical to that for the Rarest-First Model (cf. Equations 9 and 10). 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 11 and 12).

$$\overline{x} = \lambda \left[\frac{1}{UC} - \frac{1}{\mu}\right] \qquad \overline{y} = \frac{\lambda}{\mu} \qquad T = \frac{1}{UC} - \frac{1}{\mu}. \tag{29}$$

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

While it is easy to see that In-Order (FCFS), given sufficient seed capacity, can improve both sequential progress and average download latency, it should be noted that low piece diversity can degrade performance under other scenarios. For example, in scenarios without peer seeding, the rate at which peers can obtain the last piece of the file will always be limited by the rate at which the server can serve the piece (since no other downloaders can provide this piece). A more detailed discussion of the impact of the order in which peers are served using In-Order policies is provided in [30].

F. Probabilistic Policies: Portion and Zipf

Similar to the analysis of the strict In-Order policy, we consider the average behavior of the system in steady-state, and disregard details of the end game. Again, considering a specific peer that has been in the system for time t_m, we note that such a peer always could request download connections from any older peer (age $t > t_m$) or from the seeds. From any younger peer (age $t < t_m$) such a peer could request download connections with a probability $Q(t_m)$; i.e., the probability that such a peer has at least one piece that the peer wants. Given this notation, the total supply of upload connections available for this peer is:

$$\tilde{U}(t_m) = (x + y - \lambda t_m(1 - Q(t_m)))U. \tag{30}$$

As in our In-Order analysis, we use a constant α, which takes into acount the portion of the total demand xD that can be ignored.[6] Using our previous notation, the probability that a downloader of age t gets a download connection can now be approximated as:

$$p(t) = \frac{\tilde{U}(t)}{\tilde{D}(t)} = \alpha \frac{(x + y - \lambda t(1 - Q(t)))U}{xD}. \tag{31}$$

Using the above expression we can now express the average download rate γ of a downloader as:

$$\begin{aligned}
\gamma &= \frac{1}{T}\int_0^T Dp(t)Cdt \\
&= \alpha UC \left[\left(\frac{x+y}{x}\right) - \frac{\lambda}{xT}\int_0^T tdt + \frac{\lambda}{xT}\int_0^T tQ(t)dt\right] \\
&= \alpha UC \left[1 + \frac{y}{x} - \frac{1}{2} + \frac{1}{T^2}\int_0^T tQ(t)dt\right] \\
&\approx \alpha UC \left[1 + \frac{y}{x} - \frac{1}{2} + \frac{1}{K^2}\sum_{k=1}^{K} kQ(k)\right].
\end{aligned}$$

For the final step of this derivation (in which we transform an integral in the time domain to a summation in the piece domain), we make the simplifying assumption that pieces are retrieved at roughly constant rate.

For simplicity, let us introduce a constant $\beta = 1 - \frac{1}{2} + \frac{1}{K^2}\sum_{k=1}^{K} kQ(k)$. Clearly, $\frac{1}{2} \le \beta \le 1$, with $\beta = \frac{1}{2}$ corrsponding to in-order piece selection (as with $\theta \to \infty$ or the portion constant $p = 1$) and $\beta \approx 1$ corresponding to using less aggresive piece slection policies (such as Rarest-First, Zipf with a small θ, or portion with a small p) for which the piece diversity is higher.

With this notation, we can now express the steady state equations of the system as:

$$\lambda - \alpha(\beta x + y)UC = 0, \tag{32}$$

$$\alpha(\beta x + y)UC - \mu y = 0. \tag{33}$$

Solving for the average number of downloaders, seeds, and download time, we obtain the following results:

$$\begin{gathered}
\overline{x} = \frac{1}{\beta}\lambda\left[\frac{1}{\alpha UC} - \frac{1}{\mu}\right]; \qquad \overline{y} = \frac{\lambda}{\mu} \\
T = \frac{\overline{x}}{\lambda} = \frac{1}{\beta}\left[\frac{1}{\alpha UC} - \frac{1}{\mu}\right]
\end{gathered} \tag{34}$$

As expected, these equations show that a larger β (greater piece diversity) improves download times.

We next show how β can be approximated for the Zipf policy and the Portion policy. First, we must calculate the sum $\sum_{k=1}^{K} kQ(k)$. For this purpose, we need to calculate the probability $Q(k)$ that a peer with k pieces can download pieces from a peer with fewer than k pieces. Under our assumption of roughly constant download rate, this probability can be calculated as:

$$Q(k) = \frac{1}{k}\sum_{k'=0}^{k-1} Q(k', k), \tag{35}$$

where $Q(k', k)$ is the probability that a peer with k' pieces has at least one piece needed by another peer with k pieces. Utilizing our analysis of the sequential progress, these probabilities can be calculated as:

$$Q(k', k) = \sum_{j'=0}^{k'}\sum_{j=0}^{k} P(j', k')P(j, k)\phi(j', j, k', k), \tag{36}$$

[6]While α may be policy dependent, we note that α typically is very close to one, using both the Zipf and the portion policy.

where

$$\phi(j',j,k',k) = \begin{cases} 1, & \text{if } j' > j \\ 1 - (\frac{j-j'}{K-j'} + \frac{k-j}{K-j'})^{(k'-j')}, & \text{otherwise} \end{cases} \quad (37)$$

G. Model Extensions

Our detailed models can also be extended to capture retrieval time variability, TCP throughput, and peer heterogeneity [25]. In this section, we briefly describe two such extensions: one for peers who abort downloads early, and one for TCP effects. While the extensions are discussed in the context of Random and/or In-Order policies, they also apply to other piece selection policies.

We first consider partial downloaders, who consume swarm resources, but depart prematurely before download is complete. More specifically, assume that there are three types of peers: complete downloaders (x_1), partial downloaders (x_2), and seeds (y). The combined arrival rate of peers is λ, where a fraction δ are partial downloaders, and these peers only download a fraction ϕ of the file pieces. Furthermore, we assume that peers are in steady-state, and the two classes of peers on average see similar download performance.

We derive a set of five equations to characterize this system model. First, based on our assumptions, the ratio of the download times of complete and partial downloaders must be $\frac{T_2}{T_1} = \phi$. Now, using Little's Law for each of the classes, we can obtain two more equations for the number of peers of each type; i.e., $x_1 = (1-\delta)\lambda T_1$ and $x_2 = \delta\lambda T_2$. Furthermore, the seed departure rate is also equal to the arrival rate of complete downloaders. Thus, $y = \frac{(1-\delta)\lambda}{\mu}$. The fifth and final equation can be obtained by observing that the download rate ($\frac{1}{T_1} = \frac{\phi}{T_2}$) of the downloaders is equal to $\frac{(x_1+x_2+y)UC}{x_1+x_2}$. Using these five independent equations, we can now solve for the five unknowns: x_1, x_2, y, T_1, and T_2.

Solving the above equation system we obtain:

$$T_1 = \frac{1}{\phi}T_2 = \frac{1}{UC} - \frac{1-\delta}{1-(1-\phi)\delta}\frac{1}{\mu}. \quad (38)$$

As expected, this expression reduces to the previous (single class) model when $\delta \to 0$ (complete downloaders only), or $\phi \to 0$ (partial downloaders leave immediately and never affect the system behavior). When $\phi \to 1$, on the other hand, the download times are at their longest (with $T_1 = \frac{1}{UC}$), since there are no peers willing to seed. Finally, we note that the seed bandwidth decreases linearly with δ when partial downloaders download a complete file, but do not cooperate as a seed (i.e., $\phi \to 1$). For this case, we have $T_1 = \frac{1}{UC} - (1-\delta)\frac{1}{\mu}$.

Similar multi-class fluid models have previously been proposed in the contexts of seed incentives [4] and peer heterogeneity [8]. However, neither of these works considers partial downloaders. More detailed extensions are also possible for cases with lower request rates [4].

Finally, we consider TCP effects. Thus far, we have assumed perfect link utilization, using a fluid assumption for our analysis. We now show how this model can be generalized to take TCP effects into consideration. In particular, we use analytic expressions for TCP throughput, developed by Parvez *et al.* [24], to replace the capacity terms (C) in the foregoing equations. These models have the form:

$$C' = \frac{1/p}{\sqrt{\frac{2}{3p}}R + D_{FR}(W,m)}, \quad (39)$$

where p and R are the loss event rate and round-trip time, respectively, and $D_{FR}(W,m)$ is used to capture the average fast recovery period of the TCP version of interest. See [24] for details.

V. 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:

$$\int_0^t DCp(g)dg \geq (1-\varepsilon)(t-\tau)r. \quad (40)$$

Let us first consider the case of In-Order (naive) download with random peer selection. Substituting $p(g)$ from Equation 31 provides:

$$\alpha UC\left[(1+\frac{y}{x})t - \frac{\lambda}{2x}t^2\right] \geq (1-\varepsilon)(t-\tau)r. \quad (41)$$

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:

$$\tau \geq \max_{\forall t}\left[\frac{1}{MC}, t - \frac{\alpha UC((1+\frac{y}{x})t - \frac{\lambda}{2x}t^2)}{(1-\varepsilon)r}\right]. \quad (42)$$

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:

$$\tau_{min} = \max\left[\frac{1}{MC}, 2\left(\frac{1}{\alpha UC} - \frac{1}{\mu}\right) - \frac{1}{(1-\varepsilon)r}\right]. \quad (43)$$

For the In-Order (naive) policy, we have the following insights:

- 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:

$$\tau \geq \max_{\forall t} \left[\frac{1}{MC}, t - \frac{UC(1+\frac{y}{x})t}{(1-\varepsilon)r} \right]. \qquad (44)$$

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:

$$\tau_{min} = \max \left[\frac{1}{MC}, \left(\frac{1}{UC} - \frac{1}{\mu} \right) - \frac{1}{(1-\varepsilon)r} \right]. \qquad (45)$$

This analysis of In-Order (FCFS) piece selection provides the following insights:

- 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 (naive) 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.

For the Portion and Zipf policies, a more general analysis is needed. However, given the above assumptions (including separation of sequential progress and download progress), we can estimate the average startup delay that peers could have using the estimated time $T\frac{k}{K}$ to retrieve $E[j|k]$ in-order pieces (referred to as the combined media download rate). More specifically, we estimate this startup delay as:

$$\tau = \max_k \left[\frac{k}{K} T - r \frac{E[j|k]}{K} L \right], \qquad (46)$$

where r is the playback rate and L is the total file size. Here, the expression within the max brackets corresponds to the time difference between downloading k pieces, at which time we are expected to have $E[j|k]$ in-order pieces, and the time it takes to play out these $E[j|k]$ in-order pieces. Clearly, the startup delay must be at least this large for playback interruption to be avoided.

VI. Simulation Validation

In this section, we present ns-2 simulation experiments to validate our analytical models. 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 download times and startup delays are normalized to the media playback duration in the default configuration.

A. Fluid Model Validation

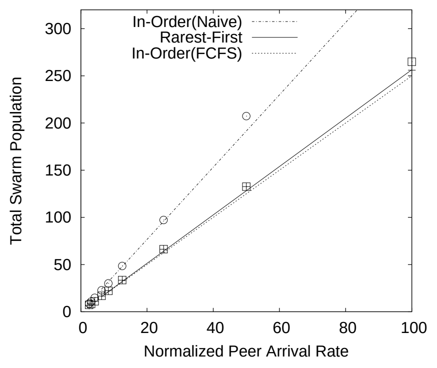
Consider first the fluid case, in which we ignore TCP effects, and peers can fully utilize their upload bandwidth whenever there is sufficient piece diversity and demand. For this case we present ns-2 fluid simulation experiments to validate the analytical models developed in Sections III, IV and V.

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.

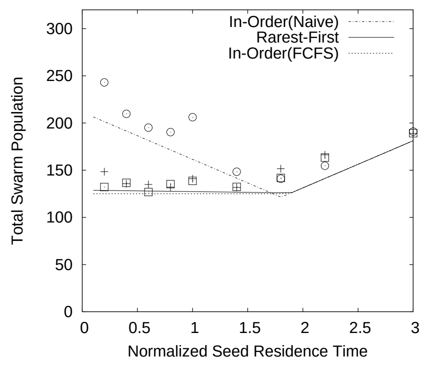
The simulation results for swarm population show good agreement with the analytical models. Figure 2(a) shows that the total swarm population is linearly dependent on the peer arrival rate, as expected. The three analytical models are presented using lines, as labeled in the graph key. The corresponding simulation results appear as points on the graph, with '+' for Rarest-First, circles for In-Order (Naive), and squares for In-Order (FCFS). In-Order (FCFS) behaves similarly to Rarest-First, while In-Order (Naive) 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 (Naive) 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 (Naive) 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.

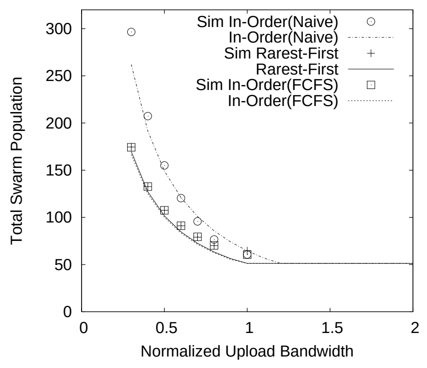
The third row of graphs in Figure 2 shows the startup delay for media playback. The analytical models predict that



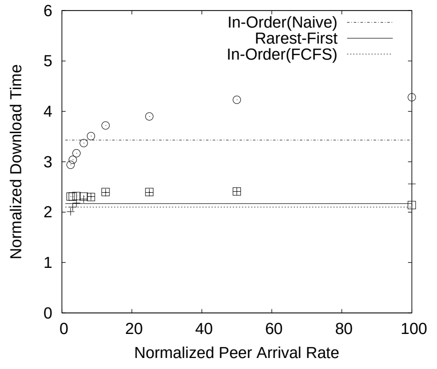
(a) Effect of Arrival Rate



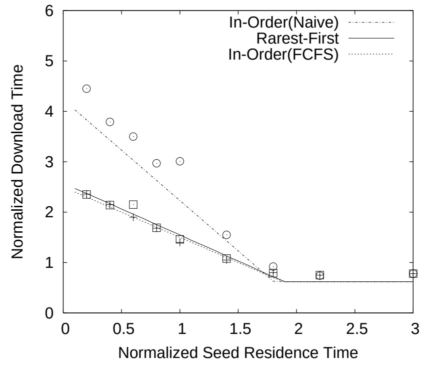
(b) Effect of Seed Residence Time



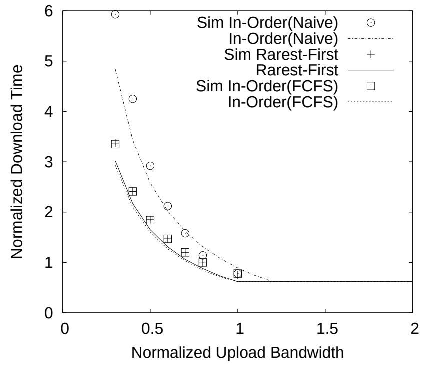
(c) Effect of Upload Bandwidth



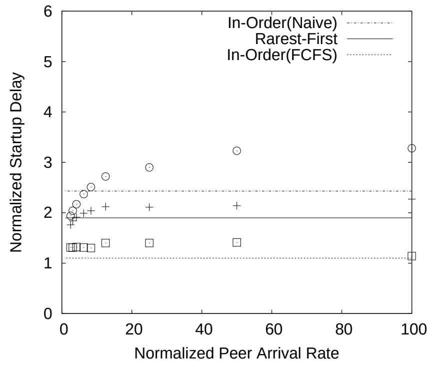
(d) Effect of Arrival Rate



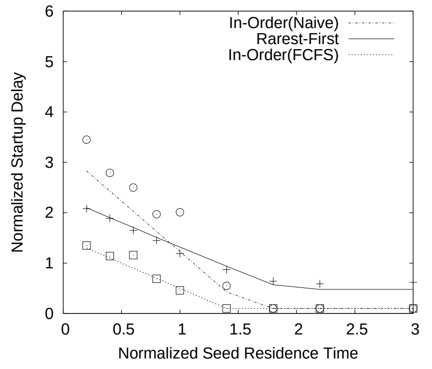
(e) Effect of Seed Residence Time



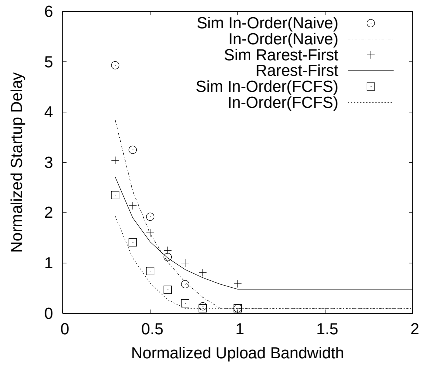
(f) Effect of Upload Bandwidth



(g) Effect of Arrival Rate



(h) Effect of Seed Residence Time



(i) Effect of Upload Bandwidth

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

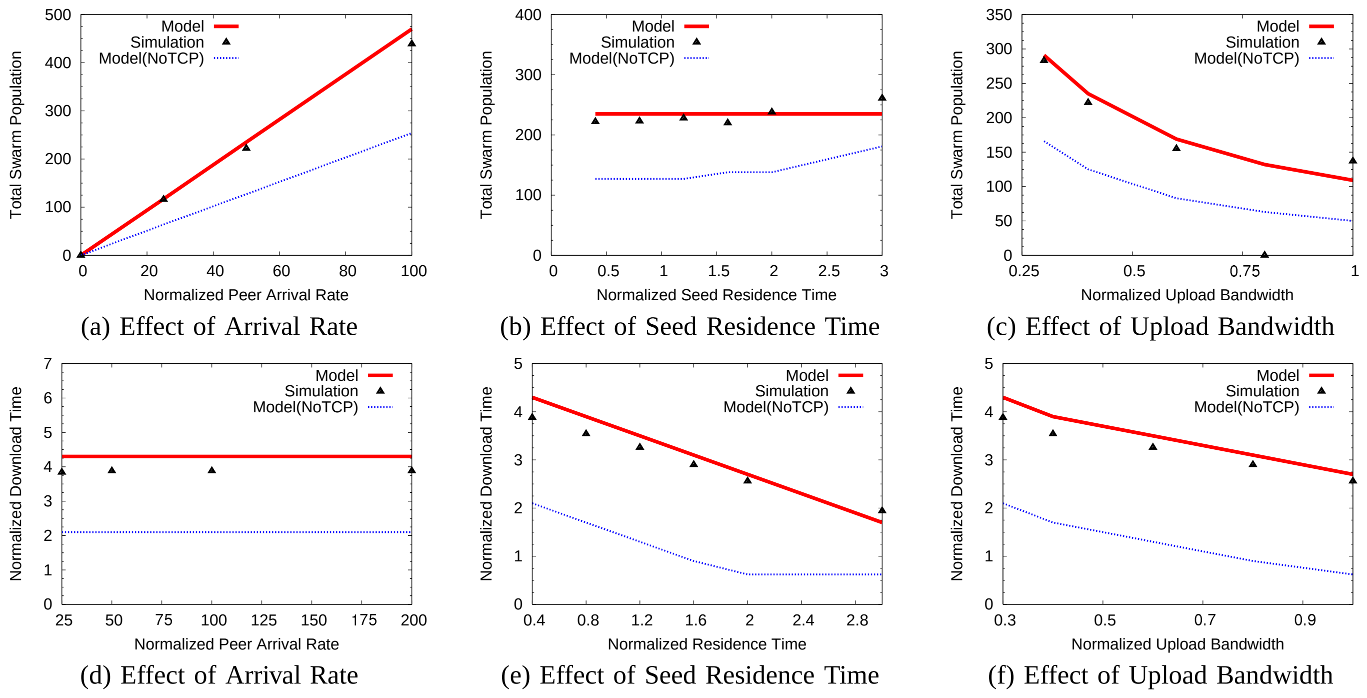
the startup delay is independent of the peer arrival rate. The simulation results in Figure 2(g) confirm this. Also, the startup delay of In-Order (FCFS) is lower than that of Rarest-First, while In-Order (Naive) is much worse. The impact of seed residence time is shown in Figure 2(h). In general, increasing 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.

B. *Packet-Level TCP Model*

We next use *ns-2* packet-level simulation experiments to validate the TCP extension of our analytical models. For simplicty we consider the Random policy, but the other experimental parameters and settings remain the same.

Figure 3 presents the results from simulation experiments for swarm population. 'Model' refers to the case of BitTorrent system prediction considering the impact of TCP throughput. 'Model(NoTCP)' refers to the case where the impact of TCP is ignored, that is, the bottleneck bandwidth is divided equally among competing flows without considering the impact of TCP behaivor.

The simulation results for swarm population show good agreement with the analytical models. Figure 3(a) shows



(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

Fig. 3. Packet-level TCP Model Validation Results (analytic results use lines; simulation results use points). Top row: Swarm Population. Middle row: Download Latency.

that the total swarm population is linearly dependent on the peer arrival rate, as expected. Figure 3(b) shows the swarm population versus the seed residence time. Figure 3(c) shows that the swarm population decreases as the upload bandwidth is increased. Note that models that consider TCP behavior predict the system quite satisfactorily, while the model without TCP is very inaccurate. This behavior justifies the use of a TCP model to explain system behavior.

The graphs in the second row of Figure 3 show 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 3(d) show a similar trend. Figure 3(e) considers the effect of seed residence time. The effect of upload bandwidth is illustrated in Figure 3(f). As expected, increasing the upload bandwidth reduces the download time, until the download bandwidth becomes the bottleneck. For all cases, the model with TCP predicts performance more accurately than the model without TCP.

VII. 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 [6], [14]. Related tree-based approaches using cache-and-relay [2], [10], [21], [27] 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 [19], [31], [32], 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 [12], 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 [7]. In this paper, we 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 [8], [13], [26]. 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 [28], characteristics related to the user behavior and peer selection strategies [20], [22], and the minimum time it takes to distribute the file from a server to a set of leechers [16]. 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 [15] and a probabilistic model [33] 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 [17], [20]) 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 system population.

Other works have shown that policies that takes the playback deadlines of each request into account (serving the earliest deadlines first, for example) can achieve even better overall user-perceived performance [5], [30]. In particular, they show that our original conjecture about the MSP optimality of In-Order in [23] was incorrect. Yang *et al.* [30] provides a detailed discussion about the impact of peer selection (as used by BitTorrent's unchoke rule, for example), when using In-Order piece selection policies.

VIII. Conclusions

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

Our models provide insight into the behavior of a P2P network used for on-demand streaming. 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 also provide analysis of probabilistic piece selection policies which provides more of a balance between in-order requirements and piece diversity.

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 using BT-like protocols in P2P networks.

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] N. Carlsson and D. L. Eager. Modeling Priority-based Incentive Policies for Peer-assisted Content Delivery Systems. In *Proc. IFIP/TC6 Networking '08*, pages 421–432, Singapore, May 2008.

[5] N. Carlsson, D. L. Eager, and A Mahanti. Peer-assisted On-demand Video Streaming with Selfish Peers. In *Proc. IFIP/TC6 Networking '09*, pages 586–599, Aachen, Germany, May 2009.

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

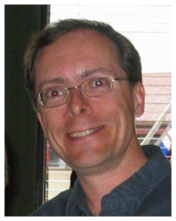
[11] A.-T. Gai, F. Mathieu, F. de Montgolfier, and J. Reynier. Stratification in P2P Networks: Application to BitTorrent. In *Proc. ICDCS '07*, Toronto, Canada, June 2007.

[12] 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.

[13] 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.

[14] 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.
[15] 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.
[16] 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.
[17] 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.
[18] 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.
[19] 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.
[20] 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.
[21] 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.
[22] L. Massoulie and M. Vojnovic. Coupon Replication Systems. In *Proc. ACM SIGMETRICS '05*, pages 2–13, Banff, Canada, June 2005.
[23] N. Parvez, C. Williamson, A. Mahanti, and N. Carlsson. Analysis of BitTorrent-like Protocols for On-Demand Stored Media Streaming. In *Proc. ACM SIGMETRICS '08*, pages 301–312, Annapolis, MD, June 2008.
[24] N. Parvez, A. Mahanti, and C. Williamson. An Analytical Throughput Model for TCP NewReno. In *IEEE/ACM Transactions on Networking*, Vol. 18, No. 2, pp. 448-461, April 2010.
[25] N. Parvez. TCP Modeling and Its Application. *PhD thesis*, University of Calgary, August 2009.
[26] 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.
[27] 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.
[28] 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.
[29] 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.
[30] Y. Yang, A. L. H. Chow, L. Golubchik, and D. Bragg. Improving QoS in BitTorrent-like VoD Systems. In *Proc. IEEE INFOCOM '10*, San Diego, CA, March 2010.
[31] 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.
[32] 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.
[33] 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.

Nadim Parvez completed his Ph.D. in the Department of Computer Science at the University of Calgary in 2009. He holds a B.E. in Computer Science and Engineering from the Bangladesh University of Engineering Technology and a M.Sc. in Electrical and Computer Engineering from the University of Manitoba. His research interests include Internet protocols, TCP modeling, peer-to-peer systems, and media streaming systems.



Carey Williamson is a Professor in the Department of Computer Science at the University of Calgary, where he holds an iCORE Chair in *Broadband Wireless Networks, Protocols, Applications, and Performance*. He has a B.Sc.(Honours) in Computer Science from the University of Saskatchewan, and a Ph.D. in Computer Science from Stanford University. His research interests include Internet protocols, wireless networks, network traffic measurement, network simulation, and Web performance.

Anirban Mahanti is a Senior Researcher at NICTA. He holds a B.E. in Computer Science and Engineering from the Birla Institute of Technology (at Mesra), India, and a M.Sc. and a Ph.D. in Computer Science from the University of Saskatchewan. His research interests include network measurement, TCP/IP protocols, performance evaluation, and distributed systems.

Niklas Carlsson is an Assistant Professor at Linköping University, Sweden. He received his M.Sc. degree in engineering physics from Umeå University, Umeå, Sweden, and his Ph.D. in computer science from the University of Saskatchewan, Canada. He has also been working as a Postdoctoral Fellow at the University of Saskatchewan, Canada, and as a Research Associate at the University of Calgary, Canada. His research interests are in the areas of design, modeling, and performance evaluation of distributed systems and networks.