Capacity Management of Seed Servers in Peer-to-Peer Streaming Systems With Scalable Video Streams

Kianoosh Mokhtarian, *Student Member, IEEE*, and Mohamed Hefeeda, *Senior Member, IEEE*

***Abstract*—To improve rendered video quality and serve more receivers, peer-to-peer (P2P) video-on-demand streaming systems usually deploy seed servers. These servers complement the limited upload capacity offered by peers. In this paper, we are interested in optimally managing the capacity of seed servers, especially when scalable video streams are served to peers. Scalable video streams are encoded in multiple layers to support heterogeneous receivers. We show that the problem of optimally allocating the seeding capacity to serve scalable streams to peers is NP-complete. We then propose an approximation algorithm to solve it. Using the proposed allocation algorithm, we develop an analytical model to study the performance of P2P video-on-demand streaming systems and to manage their resources. The analysis also provides an upper bound on the maximum number of peers that can be admitted to the system in flash crowd scenarios. We validate our analysis by comparing its results to those obtained from simulations. Our analytical model can be used by administrators of P2P streaming systems to estimate the performance and video quality rendered to users under various network, peer, and video characteristics.**

***Index Terms*—Analytical models, peer-to-peer streaming, resource allocation, scalable video streaming.**

I. Introduction

PEER-TO-PEER (P2P) and peer-assisted streaming systems have emerged as promising approaches for delivering multimedia content to large-scale user communities [1]–[3], [27]. In these systems, peers contribute bandwidth and storage to serve other peers. Since the contributions from peers are often less than the capacity needed to serve high-quality streams, a number of dedicated servers are usually deployed to boost the streaming capacity. These servers are referred to as seed servers.

In current P2P streaming systems, a video is encoded at a certain bitrate, typically ranging from 300 kbps to 1 Mbps [6]. In order to support a wider range of receivers, it is preferred to encode and serve a lower-bitrate video, but this will provide a low quality for everyone. One solution to overcome this problem is to encode and distribute multiple versions of each video, which is called *simulcasting*. However, in this approach a video has to be encoded many times for different combinations of decoding, downloading and viewing capabilities of receivers. Moreover, switching among versions is not easy, because: (i) for every switching, a client has to wait for the next Intra-coded frame of the new version, and (ii) streams of different versions could be asynchronous [12]. In addition, P2P streaming with multiple versions of the same video divides peers into separate networks which may result in reduced connectivity and less efficient utilization of peers' resources. Alternatively, multiple description coding (MDC) can encode a video into multiple descriptions, where the quality of the video is proportional to the number of descriptions received. However, MDC techniques have considerable bitrate overhead and are computationally complex [12]. In contrast, a multi-layer scalable video stream can be encoded once and a wide range of heterogeneous clients can decode it. In addition, heterogeneous clients receiving different layers can still share common layers and participate in the same overlay network, leading to a larger pool of resources. Furthermore, scalable coding has lower overhead and is simpler than MDC. Recent scalable video coding techniques, e.g., H.264/SVC, have further improved this coding efficiency and significantly outperformed previous scalable videos [20], which made them highly favorable for adoption in practice [7], [19].

In this paper, we consider P2P streaming systems that: (i) deploy seed servers to complement and boost the capacity contributed by peers, and (ii) serve scalable video streams to support a wide range of heterogeneous receivers. We are interested in managing the capacity offered by seed servers and analytically understanding how this capacity affects the performance observed by peers. We start by focusing on the problem of efficiently allocating the resources of seed servers to requesting peers according to their demands and contributions. This allocation plays a critical role for providing a *high-quality* streaming service, as we show in the paper. This is because the upload bandwidth of peers is often far less than their demanded streaming rates. For example, an average-to-good quality video stream requires about 1–2 Mbps, whereas the average upload capacity of home users with DSL and cable connections is often less than a few hundred kbps. Furthermore, careful seeding of different video substreams to peers with different demands and contributions has a significant effect on the utilization of the upload capacities of the peers themselves. Then, we develop an analytical model to study the dynamic behavior of P2P streaming systems with scalable video streams. Our model aims at forecasting this behavior and answering a number of key questions regarding the streaming system.

Manuscript received January 26, 2012; revised April 23, 2012; accepted May 28, 2012. Date of publication October 16, 2012; date of current version December 12, 2012. This work was supported in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada and in part by the British Columbia Innovation Council. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Feng Wu.

K. Mokhtarian was with the School of Computing Science, Simon Fraser University, Surrey, BC V3T 0A3, Canada, and is now with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON M5S 3G4, Canada (e-mail: kianoosh.mokhtarian@mail.utoronto.ca).

M. Hefeeda is with the School of Computing Science, Simon Fraser University, Surrey, BC V3T 0A3, Canada.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TMM.2012.2225042

In particular, the contributions of this paper can be summarized as follows.

- We formulate the problem of optimally allocating the seeding capacity to peers requesting scalable videos. We show that this problem is NP-complete.
- We propose an approximation algorithm to solve the seeding capacity allocation problem. We evaluate the proposed algorithm analytically and in a simulated P2P streaming system. The results confirm the near-optimality of the proposed algorithm, and show that higher-quality videos are delivered to peers if it is employed for allocating seed servers.
- We propose an analytical model to forecast the performance of P2P on-demand streaming systems serving scalable videos and using our seed server allocation algorithm. Our analysis takes as input the characteristics of the network, including the distribution of upload and download bandwidths of peers, peer arrival and failure rates, and video bitrates. Our analysis is general and can be applied to P2P streaming systems with different characteristics and network conditions. We present numerical analysis of example systems and derive insights on their performance. In addition, we validate our analytical results through simulations.
- We show how our analytical model can be used to answer the following important questions:

  Q_1. How can administrators of P2P streaming systems optimally utilize the capacity of seed servers?

  Q_2. What is the expected throughput (total bitrate served) of a given P2P streaming system? The throughput directly impacts the video quality delivered to peers.

  Q_3. To increase the throughput to a desired level, how much seeding capacity is needed?

  Q_4. How many peers a system can support in case of a flash crowd arrival of peers?

  Q_5. How effective are scalable video streams in P2P streaming systems compared to nonscalable streams for supporting flash crowds?

This paper is organized as follows. The related works are summarized in Section II. In Section III, the seed capacity allocation problem is defined and proven to be NP-complete. In the same section, we present the proposed approximation algorithm. In Section IV, we present our analytical model for P2P streaming systems that uses the approximation algorithm developed in Section III. In Section V, we conduct an analysis of the analytical model on a sample P2P network. Then, we validate our analysis using simulations, and we use the analysis to study the performance of P2P streaming systems in Section VI. Section VII concludes the paper.

II. Related Work

The problem of peer-to-peer streaming has been studied by numerous previous works from various angles, such as overlay construction, distribution of data availability information, and piece scheduling algorithms. Nevertheless, the problems we study in this paper, resource allocation and theoretical analysis of the behavior of these systems, have received less attention in the literature. Moreover, most previous works in this area consider streaming of nonscalable video streams only.

A. P2P Streaming With Scalable Videos and Seed Servers

Cui *et al.* [4] and Rejaie *et al.* [18] study P2P streaming systems with scalable videos, focusing on the tasks of peers. An algorithm is presented in [4] to be run on each peer independently to decide how to request video layers from a given set of heterogeneous senders, assuming layers have equal bitrate and provide equal video quality. Hefeeda *et al.* [5] study this problem for fine-grained scalable (FGS) videos, taking into account the rate-distortion model of the video for maximizing the perceived quality. We too consider video layers with heterogeneous rates and quality enhancements. In the framework presented in [18], the problem of requesting from a set of senders is studied. Hu *et al.* [8] design a taxation mechanism for fairness among peers with diverse download and upload bandwidths requesting a scalable video stream. Lan *et al.* [11] present a high-level architecture for data-driven P2P streaming with scalable videos. Packet scheduling strategies for downloading scalable videos are studied in [23]. All of these works do not consider the functionalities of seed servers. Xu *et al.* [24] study the functionality of seed servers for P2P streaming. However, their work is only for nonscalable video streams, and they assume that peers' upload bandwidth can only take power of 2 bitrates. The case for scalable video streams is more challenging as various substreams need to be handled. In [4], seed servers are assumed to always have enough capacity to serve all requests, which is not realistic. We consider a more practical scenario in which seed servers have finite capacity, and this finite capacity needs to be optimally allocated to requesting peers such that a higher-quality video is delivered to all peers.

B. Analytical Models for P2P Streaming Systems

Current analytical works for studying the behavior of P2P streaming systems assume that the video streams are encoded in nonscalable manner. Nonscalable video streams have fixed bitrates and cannot be adapted easily. For example, Tu *et al.* [22] present an analytical framework for studying the performance of P2P streaming systems. This framework mathematically analyzes the pattern of capacity growth of a given dynamic P2P system over time. It then estimates the time at which the servers can be turned off. A similar analysis is done in [10] based on a stochastic fluid model, which finds the maximum streaming bitrate that can be supported in a P2P system. However, the works in [10] and [22] only consider nonscalable streams with a constant bitrate. They also divide peers into a number of classes based on their bandwidth contributions. In addition, in [10] only live streaming scenarios are considered. The work in [22] can analyze on-demand streaming scenarios as well, but the time granularity of this analysis, i.e., the length of the smallest time unit, is the length of a complete streaming session. In our analytical model, we consider on-demand streaming of scalable video streams with variable bitrates, and we do not make any restrictive assumptions on the bandwidth of peers. Moreover, we analyze the capacity of the system over time with a fine granularity, which is the length of a video segment of a few minutes at most. This enables us to capture the dynamics of the network

more accurately. For example, the time a peer stays in the network for seeding is often a fraction of the video length, which cannot be captured when we analyze the system based on a time unit equal to a complete video length.

Liu *et al.* [13] analyze peer-assisted streaming systems by constructing a tree structure for the network and provisioning resources for it. In contrast, we do not assume any particular structure imposed on the network. That is, each peer can decide its neighbor list, based on the information it receives from trackers and/or other peers. Thus, our approach can be used to analyze mesh-based and receiver-driven systems. Since mesh-based systems are employed by most of today's P2P streaming systems [1], [2], [6], [27], our analytical model is more practical and useful than previous ones. Another performance analysis of P2P streaming systems is conducted by Small *et al.* in [21]. However, similar to [13], the authors consider constructing a specific P2P topology for the overlay network, unlike our analysis which does not make any particular assumption on the overlay topology. Xu *et al.* [25] present an analytical model to assess the scalability and capacity growth of a P2P streaming system. They assume that peers leave the system only after contributing some threshold capacity. Yin *et al.* [26] conduct a similar study where peers may leave at random. However, the authors only estimate a scaling factor for the system, which is the fraction of the peers directly served by dedicated servers, as a function of the average peer upload bandwidth. Zhou *et al.* [28] and Parvez *et al.* [17] analyze the performance of P2P streaming systems when two different chunk selection strategies are employed by peers: rarest-first and sequential. However, the work in [28] only analyzes live streaming scenarios, and the one in [17] assumes that all peers have equal bandwidth and all P2P connections have the same throughput. Moreover, in all these works a nonscalable stream (often at a single bitrate) is considered. Our model considers an adaptable layered stream, and it takes into account the distribution of peer demands and upload bandwidths to estimate the video quality delivered to peers and the required server bandwidth for achieving any desired level of quality.

Finally, we mention that preliminary parts of this work appeared in [15] and [16]. We developed an analytical model for P2P streaming systems [16], and designed algorithms for the seed allocation problem [15]. In this paper, we extend our previous works and present a unified framework for addressing the capacity management problem in P2P streaming systems. We extend our server allocation algorithm to operate in real-time with no queuing delay, and with lower computation overhead. In addition, we present a scalable P2P architecture aligning our resource allocation algorithm and our analytical model, and demonstrate its feasibility for large-scale systems. Moreover, we conduct more experiments to analyze the advantages of P2P streaming with scalable videos for handling flash crowds, compared to nonscalable streaming used in most of today's P2P streaming systems [1], [2], [6], [27].

III. Seeding Capacity Allocation

In this section, we study the problem of allocating the resources of seed servers. As discussed earlier, these servers are a key component of the P2P streaming system when a high-quality streaming service to peers with diverse (and often limited) upload capacities is desired. We consider allocating the seeding resources to peers such that a system-wide utility function is maximized, e.g., average video quality served to users. We first present the considered P2P system model, and we formally state the seeding capacity allocation problem. This is followed by an approximation algorithm to solve this problem, **which answers question $\mathbf{Q}_1$ discussed in Section I**. We then discuss the feasibility of the considered P2P architecture and the proposed algorithm for large-scale streaming systems.

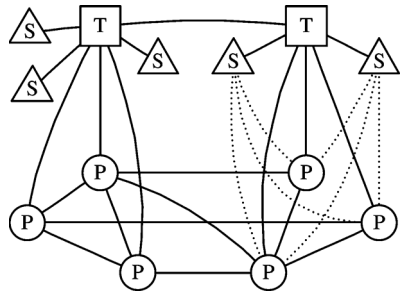


Fig. 1. The considered P2P streaming model. T, S, and P represent trackers, seed servers, and peers, respectively.

A. System Model and Problem Statement

The considered P2P streaming architecture is illustrated in Fig. 1 and consists of peers, seed servers, and trackers. Peers join the system by contacting one of the trackers, to which they send their subsequent requests. Each tracker controls a number of seed servers. A tracker receives periodic update reports from its peers, informing it about their available data and capacity. This enables the tracker to monitor its network and keep track of the set of active peers, their contributions, and their data availability. Note that trackers do not keep track of the topology of the network, i.e., the list of partners of each peer. Trackers exchange periodic messages (e.g., once every minute) to update their system-wide data availability information, as described in Section III-D. Trackers allocate the resources of their seed servers to peers' requests. That is, the tracker determines the requests to be served by one of the seed server and those to be forwarded to other peers in the system. Peers download the video in a streaming form, meaning that the video is downloaded sequentially at a bitrate equal to the bitrate of the video. Peers serve lower layers of the videos first, in order to avoid the situation that some peers are starving (for not receiving lower layers) while other peers are receiving the highest quality.

Let V denote the set of video files (a list of notation is provided in Table I for quick reference). We divide a video into short intervals, called video segments. A video segment is considered an atomic unit of adaptation, meaning that the number of layers received by a peer is assumed constant during a segment, but may vary between consecutive segments. Let us assume for now that the tracker queues the requests received from peers, and solves the allocation problem for existing requests every few seconds. We then extend the algorithm to perform continuously in real-time as the requests arrive.

Assume there are K requests in the queue when running the algorithm. Each request req_k is in the form

TABLE I
LIST OF NOTATIONS USED IN THIS PAPER. BOLD SYMBOLS REPRESENT RANDOM VARIABLES

Parameter	Description	Section
C	Capacity of seed servers (bps).	III, IV
u_p	Upload capacity (bps) of peer p.	III, IV
$\mathbf{D}, \mathbf{U}, \mathbf{U}_l$	Random variables representing the download and upload bandwidth of peers. $\mathbf{D}$ and $\mathbf{U}$ are not independent. $\mathbf{U}_l$ represents the upload bandwidth of peers for layer l of the video.	IV
$\mathbf{N}$	Random variable representing the number of arrivals per segment length.	IV
$\gamma(t)$	The probability that a peer that is at segment t fails/leaves.	IV
P_s	Set of peers watching segment s of the video.	III, IV
S, L	Number of segments and number of layers of the video.	III, IV
T_{seg}	Video segment length (in seconds).	III, IV
T_{seed}	The expected time a peer seeds a video file after watching.	IV
$r_{v,l}$	Average rate (bps) of layer l of video v.	III
$q_l(t)$, $r_l(t)$	Video quality added by and the bitrate of the l-th layer at segment t.	IV
K, n_k	Number of requests in the queue, and number of layers in request k.	III
req_k, $req_{k,j}$	k-th request in queue, and the sub-request of it asking for j lowest layers.	III
$c_{k,j}$, $b_{k,j}$	Cost and utility (benefit) of sub-request $req_{k,j}$.	III
$e[l], o[l]$	The capacity served for layer l *to* and *by* peers at a given video segment.	IV
$f(e, l, r_l, n)$	The capacity that can be served for layer l (at rate r_l) by n peers demanding it at a segment, when an input capacity e is served to them.	IV

$\{req_k.p,\ req_k.v,\ req_k.t,\ req_k.l_1,\ req_k.l_2\}$, meaning that peer $req_k.p$ is requesting layers $req_k.l_1$ to $req_k.l_2$ (inclusive) of video stream $req_k.v$, starting at segment $req_k.t$; the peer could be receiving layers 1 to $req_k.l_1 - 1$ from other peers. Since req_k is requesting for $n_k = req_k.l_2 - req_k.l_1 + 1$ layers and may be admitted partially, we break it to n_k *sub-requests*, denoted by $req_{k,j}$ $(1 \le j \le n_k)$. A sub-request $req_{k,j}$ represents a request for the j lowest requested layers, i.e., $req_{k,j}$ corresponds to layers $req_k.l_1$ through $req_k.l_1 + j - 1$. This way of breaking a request into sub-requests ensures that no invalid subset of layers is served. Let $r_{v,l}$ denote the bitrate of the lth layer of video v, and u_p the upload capacity of peer p.

Serving each sub-request $req_{k,j}$ has a cost $c_{k,j}$ for the seed server, which is the sum of the bitrates of the j requested layers. Letting v simply denote the requested video in req_k, we denote the costs of req_k's sub-requests by:

$$c_{k,j} = \sum_{l=req_k.l_1}^{req_k.l_1+j-1} r_{v,l} \quad (1 \le k \le K,\ 1 \le j \le n_k). \tag{1}$$

Moreover, by admitting $req_{k,j}$, a utility (benefit) $b_{k,j}$ is gained by the system, which consists of the utility of serving the associated layers to the corresponding peer, that is, $\sum_{l=req_k.l_1}^{req_k.l_1+j-1} b_{self}(req_k.p, l)$, and the utility gained when the peer shares those layers with the network, denoted by $\sum_{l=req_k.l_1}^{req_k.l_1+j-1} b_{share}(req_k.p, l)$. Our algorithm is not restricted to a specific utility function. For example, one can define the utility function in a way to maximize the average quality received by peers or to provide max-min fairness among quality received by peers according to their demands; detailed examples can be found in [14, ch. 4.4.3].

For calculating $b_{share}(p, l)$, we need to consider the peer serving those layers (or part of them) to its partners, those partners serving to their partners, and so on. Taking these neighborhood details into account requires knowledge of the full overlay topology, which is difficult to maintain for a dynamic P2P system. We therefore compute $b_{share}(p, l)$ as the *expected* utility that the system gains when a peer shares a video layer with the network. Detailed calculation of this function for the two cases of maximizing the average quality and max-min fairness are available in [14, ch. 4.4.3] and are omitted here due to space limitations. The procedure in a high level is to first estimate the expected upload rate of peer p for different video layers, as a function of the peer's demand and bandwidth. If the system objective is to maximize the average video quality over all peers, this expected rate can directly estimate $b_{share}(p, l)$ as the video quality increase brought to the system. For a more complicated objective such as max-min fairness according to the peers' demands, $b_{share}(p, l)$ is estimated as a function of the expected upload rate and the distribution of layer demand in the system [14, ch. 4.4.3].

1) Problem 1: (Seeding Capacity Allocation): Given requests $req_1, \ldots, req_K$, their costs $c_{k,j}$ bps and utilities $b_{k,j}$ $(1 \le k \le K,\ 1 \le j \le n_k)$, and a seeding capacity C bps, find the x_k $(0 \le x_k \le n_k)$ value for each req_k which indicates that sub-requests $req_{k,1}, req_{k,2}, \ldots, req_{k,x_k}$ should be served out of req_k in order to maximize the system-wide utility.

This problem is formulated as follows. Find x_k in order to:

$$\max \quad \sum_{k=1}^{K} b_{k,x_k} \tag{2a}$$

$$\text{s.t.} \quad \sum_{k=1}^{K} c_{k,x_k} \le C \tag{2b}$$

$$x_k \in \{0, 1, \ldots, n_k\} \ \ (1 \le k \le K). \tag{2c}$$

Theorem 1: The seeding capacity allocation problem defined in (2) is NP-complete.

Proof: We prove the NP-completeness by reducing the Knapsack Problem [9] to a simplified version of the seed server allocation problem. Suppose that all videos are single-layer coded and thus all requests are for the first layer. In this case, all x_i values are either 0 or 1. This special case of the problem is equivalent to the 0–1 Knapsack Problem. In addition, a solution for the seed server allocation problem can easily be verified in polynomial time. Hence, the seed server allocation problem is NP-complete. □

B. Seeding Capacity Allocation (SCA) Algorithm

In this section, we present an approximation algorithm for the seed server allocation problem, which is to be executed by the tracker to decide which requests should be served among those currently pending in the queue. Our approximation is based on relaxing the Integer Programming (IP) problem in (2) to its equivalent Linear Programming (LP) problem as in (3). That is, we now allow a layer to be partially served, even though it is not meaningful in practice.

$$\max \quad \sum_{k=1}^{K} \left(b_{k,\lfloor x'_k \rfloor} + \{x'_k\}(b_{k,\lceil x'_k \rceil} - b_{k,\lfloor x'_k \rfloor}) \right) \tag{3a}$$

$$\text{s.t.} \quad \sum_{k=1}^{K} \left(c_{k,\lfloor x'_k \rfloor} + \{x'_k\}(c_{k,\lceil x'_k \rceil} - c_{k,\lfloor x'_k \rfloor}) \right) \leq C \tag{3b}$$

$$x'_k \in \mathbf{R} \tag{3c}$$

$$0 \leq x'_k \leq n_k \ \ (1 \leq k \leq K), \tag{3d}$$

where $\{x'_k\} = x'_k - \lfloor x'_k \rfloor$ denotes the fractional part of x'_k.

Having solved the LP problem in (3) and obtained the x'_k values, we obtain a valid solution to the original IP problem by rounding down all x'_k values: $x_k = \lfloor x'_k \rfloor$. Clearly, x_k values form a valid answer for the IP form in (2), since they satisfy both constraints (2b) and (2c). We will see shortly that after this relaxation and down-rounding, how close the objective function (2a) will be to the optimal solution.

The proposed seeding capacity allocation (SCA) algorithm is shown in Fig. 2. Sub-requests are sorted in decreasing order of utility-to-cost ratio and are picked one by one as long as the total seeding capacity allows. Note that some sub-requests are overlapping, e.g., sub-request $(k, 1)$ is a subset of sub-requests $(k, 2), \ldots, (k, n_k)$. Hence, when deciding to serve a sub-request (k, j), we need to check whether another sub-request of request $\#k$ is already admitted in previous iterations. This is illustrated in Lines 6–9 of the code in Fig. 2. If a sub-request in the form of $(k, j' > j)$ is already served, (k, j) can be simply skipped. If one in the form of $(k, j' < j)$ is served, we only take those layers from sub-request (k, j) that are not already served.

The following theorem shows the approximation factor of the SCA algorithm. The proof is omitted due to space limitation [15].

Theorem 2: If all costs $c_{k,j}$ are bounded as $c_{k,j} \leq c_{max}$, meaning that c_{max} is the maximum bitrate of a video stream, and assuming $c_{max} < (1/2)C$ where C is the capacity of seed servers (hence practically $c_{max} \ll C$), the SCA algorithm is a $c_{max}/C - c_{max}$-factor approximation for the seed server allocation problem, i.e., the utility obtained by the SCA algorithm is $z \geq (1 - c_{max}/C - c_{max})OPT$, where OPT is the optimal utility.

For example, for a 2 Mbps video and a seeding capacity of 25 Mbps, it is guaranteed that the SCA approximation algorithm will produce results no worse than 91% of the optimal in this case.

We should also note that the SCA algorithm automatically adjusts the serving resources with respect to different volumes of demand across video files, i.e., to video popularities. For instance, if we are serving the top 10% of the requesting peers (according to their capability to share with the system), automatically more peers are served for a more popular video. The seeding algorithm will always try to equalize, based on the given utility function, either the level of video quality received by all peers watching different videos, or the ratio of the received quality level over the demanded level for all peers [14, ch. 4.4.3].

Seed Capacity Allocation Algorithm

```
SCA(K, n[], b[][], c[][], C)
// K: number of requests, C: seeding capacity
// n[k]: number of sub-requests in the k-th requests
// b[k][j], c[k][j]: utility and cost of sub-request (k, j)
// Output:
// x[]: number of sub-requests to serve from request #k
// z: total utility gained
1.  x[] ⟵ CreateArrayOfZeros(K)
2.  z ⟵ 0
3.  S[] ⟵ all sub-requests (k, j) //a total of ∑_{k=1}^{K} n[k]
4.  Sort S[] in decreasing order of utility-to-cost ratio
5.  for (k, j) ∈ S[] do
6.      if x[k] > j then
7.          continue //subset of an already-served sub-request
8.      cost ⟵ c[k][j]−c[k][x[k]]
9.      utility ⟵ b[k][j]−b[k][x[k]]
10.     if cost ≤ C then
11.         C ⟵ C − cost
12.         z ⟵ z + utility
13.         x[k] ⟵ j
14. done
15. return x[], z
```

Fig. 2. The SCA algorithm for the seed capacity allocation problem.

The SCA algorithm consists of sorting sub-requests, which runs in $O(K' \log K')$ where $K' = \sum_{k=1}^{K} n_k$, and then performing up to K' iterations of $O(1)$. Hence, the total running time is $O(K' \log K')$.

C. The Real-Time SCA

The SCA algorithm described so far runs periodically on a queue of pending requests. To avoid delays in processing and responding of requests, specially for requests eligible for immediate serving, we improve the SCA algorithm to operate in a real-time manner. To this end, we maintain a priority queue at each tracker, to which the requests are added (and possibly immediately served) as they arrive. Recall that the SCA algorithm introduced so far is based on sorting sub-requests according to utility-to-cost ratios, and picking those with higher ratios. Thus, we maintain a max-heap of requests at each tracker, prioritized based on the utility-to-cost ratio.

From this priority queue, the first i sub-requests at the head are served by seed servers. Once a new request is received, its sub-requests are extracted and immediately inserted in the priority queue according to their utility-to-cost ratios. Depending on the obtained positions in the queue, one of the new sub-requests may be entitled for service. For example, if the new request is asking for 5 layers and 3 of the 5 new sub-request are entitled service according to their ratios, the one with the highest number of layers is picked. In turn, a previous request may stop receiving data from seed servers (i.e., the request with the lowest

utility-to-cost ratio currently being served), which instead will be forwarded to other peers in the system.

The processing for each new request, assuming it consists of l sub-requests and that there are K'' sub-requests currently being served, is the insertion (and possibly deletion) of l entries in a max-heap of size K''. This makes the processing time $O(l \log K'')$ for the given request. To further understand this time compared to the non-real-time operation of SCA, let ρ denote the average number of requests per second, $\bar{l}$ the average number of sub-requests per request, and Δ the time between two runs of the non-real-time SCA. The running time of the non-real-time SCA is of $O(K' \log K')$ per each run, where K' is the number of sub-requests pending, i.e., $K' = \rho \Delta \bar{l}$. The expected processing time per second is hence $O(K' \log K'/\Delta) = O(\rho \bar{l} \log K')$. In the real-time mode, on the other hand, SCA takes $O(l \log K'')$ per request, resulting in an average time of $O(\rho \bar{l} \log K'')$ per second. Note that K', the number of pending requests, is most often larger than K'', the number of sub-requests that can currently be served by seed servers; otherwise we could always serve all peer requests using seed servers which is far from reality. This means that the processing load of the SCA algorithm is even smaller when operating in real-time.

D. System Scalability

The P2P streaming model that we consider (Section III-A) includes one or more trackers that have a global view of data availability in the network. This is achieved through periodic update reports that peers send to their corresponding tracker, and trackers exchanging this information periodically. In this section, we briefly discuss the feasibility of this architecture for large scales using some realistic numbers.

1) Peer-Tracker Communications: Each peer notifies the tracker about the video layers that it has for each segment of the video. Thus, for each segment a small message is sent signalling the video ID (e.g., 8 bytes), peer ID (4 bytes), segment (2 bytes) and the number of layers (2 bytes, assuming state-of-the-art H.264/SVC scalable videos with multi-dimensional scalability [20]). This message along with the TC/IP packet overhead is less than 60 bytes, which is sent once in every video segment length, T_{seg}—typically ranging from 10 seconds to 2 minutes. Assuming each tracker handles 10–100 thousand peers and $T_{seg} = 30$ seconds, this yields a communication load of 20–200 kB/s received by each tracker, which is small. Upon arrival of each message, the tracker simply updates an entry in a table (discussed shortly), which is of negligible processing load.

2) Tracker Processing Overhead: As analyzed earlier, the running time of the SCA algorithm can be roughly thought of as $\rho \bar{l} \log K''$ iterations per second. Consider that each tracker is managing 1 Gbps seeding capacity, the average video has a bitrate of 2 Mbps and is encoded in 10 layers of 200 kbps each, $\bar{l} = 5$, and that there are $\rho = 2000$ requests per second arriving at the tracker; K'' equals 1 Gbps divided by the average sub-request bitrate (1 Mbps), hence $K'' = 1000$. This yields up to 100 K iterations per second, which is a small CPU load even for a commodity PC. We further note that the number of trackers can always be adjusted to manage the tracker load.

3) Tracker Memory Load: Each tracker maintains per each [peer, video] pair a list keeping track of the layers that the peer holds from each segment of the video. The number of entries in this list is the number of video segments, and each entry is at most 2 bytes long considering the most flexible stream H.264/SVC [20]. Assuming an average video length of 2 hours, $T_{seg} = 30$ seconds, a total peer population of 1 million, and that the average peer participates in the swarm of 10 videos (watching one, partially seeding others), this list is less than 500 bytes for each [peer, video] pair, 4.5–5 GB in total. This is a feasible amount of memory for a server machine handling thousands of peers.

4) Inter-Tracker Communications: Trackers exchange periodic information about data availability at their peers. Let s denote this period in number of video segment lengths (i.e., a period of sT_{seg} seconds); s is typically one or a few. The major part of the information to exchange consists of the video layers that each peer holds from each segment. Only changes since the last update, sT_{seg} seconds ago, will be signalled. To analyze the amount of this information, we note that each peer can be watching at most one stream (of S segments) at a time. Therefore, S entries are updated and need to be signalled in a window of ST_{seg} seconds, meaning an amortized amount of 1 entry per peer per T_{seg} seconds, and s entries per peer in each batch between trackers; each entry is at most 2 bytes as discussed earlier. Assuming even the conservative values of $T_{seg} = 10$ seconds and $s = 1$, and that there are 10 trackers each handling 100 000 peers, 9 data chunks of 200 KB are sent/received by each tracker at the end of a 10-second interval. This yields a total of 1.8 MB sent and received every 10 seconds, which is clearly a feasible amount.

IV. Analytical Model for Capacity Planning

In this section, we present an analytical model for forecasting the dynamic behavior of P2P streaming systems, which employs the SCA algorithm for allocation the resources of seed servers. We first present a high level overview of the analysis, followed by step-by-step details. We derive general formulas which can yield the throughput of the P2P system as a function of the characteristics of the network. These general formulas can be readily customized for a given set of network characteristics, and allow us to capture a wide range of network specifications. This mainly consists of replacing the general variables used in this section, e.g., random variables representing peer bandwidth distributions, by values specific to the P2P system being analyzed. In Section V we present an example of this customization.

A. Overview

We divide a video file into S small segments. Let P_s denote the set of peers that are currently watching the video at segment s where $1 \leq s \leq S$. A peer can only serve data to peers that are at earlier segments. In other words, a peer in P_s can be served by any peer in $P_{s'}$, $s < s' \leq S$, given that the requested layer is available at the sending peer. Peers in P_s cannot be served by any peer in $P_{s'}$ where $1 \leq s' \leq s$. In addition to receiving from other peers, a peer may request to receive the stream from seed servers. The notations used in the analysis are summarized in Table I. In the modeled system, a peer can be watching one

video stream at a time. Since a peer's upload bandwidth is typically less than its streaming rate, during a streaming session the peer will use its entire upload bandwidth for uploading pieces of the video being downloaded; this assumption, while realistic according to typical user bandwidths in the Internet, is to make the analysis feasible (not a strict must for the system). A peer that is not watching any stream can seed one or more video files. T_{seed} is the average effective seeding time for each file.

Our analysis takes the following variables as inputs, and it outputs the throughput of the network in various forms, such as the average video quality delivered to peers.

- The joint distribution of upload and download bandwidth of peers. We define random variables $\mathbf{D}$ and $\mathbf{U}$ as the download and upload bandwidth, which are not independent. Let $Pr(x_1 \leq \mathbf{D} \leq x_2, y_1 \leq \mathbf{U} \leq y_2)$ where $x_1, x_2, y_1, y_2 \in \mathbb{R}$ denote the probability that a peer's download and upload bandwidth values are in the range $[x_1, x_2]$ and $[y_1, y_2]$, respectively.
- The expected peer seeding duration T_{seed} for a file.
- The seed server capacity C, number of layers L, and bitrate of each layer $r_l(t)$ at each segment t.
- The distribution of peer arrivals. $\mathbf{N}$ is the random variable representing the number of arrivals per segment and $Pr(\mathbf{N} = n)$ denotes the probability of having n arrivals in T_{seg} seconds.
- The peer failure rate, $\gamma(t)$, which is the probability that a peer leaves at segment t. $\gamma(t)$ is a function of t to capture realistic peer behaviors, e.g., peers that have stayed longer are less likely to leave.

At a high level, the analysis proceeds as follows. Recall that a peer in P_s can stream to peers in $P_{s'} (s' < s)$ only. Accordingly, starting from the last segment of the video to the first ($s = S, S-1, \ldots, 1$), in each step of the analysis we consider the set of peers that are watching the same video segment s, i.e., peers in P_s. We estimate the output capacity of these peers as a function of the capacity served to them. Using this function, we can compute the capacity that is going to be served to these peers and the capacity that they will serve to peers in $P_1, \ldots, P_{s-1}$. We then proceed to segment $s-1$ and so on. In our analysis, we treat capacity values as arrays rather than single numbers, where $e[1], \ldots, e[L]$ represent the capacity served for layers $1, \ldots, L$ to P_s, and $o[1], \ldots, o[L]$ represent the capacities served by P_s. We will later see how to divide a single seeding capacity value C between $e[l]$ values for the layers. To analyze the set of peers at a given segment, we define the function $o = f(e, l, r_l, n)$ as the output capacity of peers for layer l when this layer is served to them, i.e., n peers demanding the layer at the segment, with an input capacity e. The bitrate of the video layer at that segment is r_l.

In the remainder of this section, we will first obtain the function $f(\cdot)$ in Section IV-B. This function captures the distribution of only one video segment to a given number of peers. We employ this function in Section IV-C to analyze the distribution of the complete video in a dynamic network.

B. Distribution of One Video Segment

We define function $o = f(e, l, r_l, n)$ as the output capacity for layer l when serving this layer with a capacity e to n peers demanding it at a segment. r_l is the bitrate of the layer at the segment. Clearly, if e is equal to or larger than the demand of n peers, i.e., $e \geq r \times n$, all the n peers can be served using e. On the other hand, if $e < r \times n$, we can serve only a subset of peers. The estimated output capacity o depends on how these subsets are chosen. We consider two cases for this purpose: (i) when the serving is done optimally which the SCA algorithm tries to do (Section III-B), and (ii) when the serving is done randomly, as in many current P2P streaming systems. We distinguish three different $f(e, l, r, n)$ functions: $f_{opt}(\cdot)$ for case (i), $f_{rnd}(\cdot)$ for case (ii), and $f_{all}(\cdot)$ for the case where $e \geq r_l \times n$, in which it does not matter how to serve peers since the demand is smaller than the input capacity.

Among these three forms, we first obtain $o = f_{opt}(e, l, r_l, n)$. For this case, peers are sorted based on the bitrate that they can stream layer l to other peers, and are selected one by one for being served with layer l. In our model, peers are expected to use their upload bandwidth for serving lower layers first, and also as many layers as they can upload. This is to avoid starvation of some peers in the system (enough lower layers served) while also getting some higher quality layers distributed in the network. Therefore, the bitrate $u_{p,l}$ at which a peer p streams layer l to other peers can be estimated as:

$$\min\{u_p - \sum_{i=1}^{l-1} u_{p,i},\ r_l\}\ (1 \leq l \leq L_p),$$

where L_p is the number of layers that peer p demands, and u_p is its total upload bandwidth. In case u_p is higher than the total bitrate of the demanded layers, the peer is able to serve more than its downloaded stream rate (layers 1 to L_p). It therefore divides the remained bandwidth from the above equation evenly among the layers:

$$u_{p,l} = \min\{u_p - \sum_{i=1}^{l-1} u_{p,i},\ r_l\} + \max\{\frac{u_p - \sum_{i=1}^{L_p} r_i}{L_p}, 0\}. \quad (4)$$

Using (4) and the distribution of upload bandwidths ($\mathbf{U}$), we can calculate the distribution of $\mathbf{U}_l$: the random variable representing the upload rate of peers for layer l.

Peers with higher $u_{p,l}$ values are selected for being served with layer l. The number of peers that can be served using capacity e is $k = \lfloor e/r \rfloor$. Since for calculating $f_{opt}(\cdot)$, we know that $k < n$, the value to be calculated, o, consists of "the total capacity that can be streamed by the k peers that have the highest $\mathbf{U}_l$-capacity values among the total n peers". We estimate this capacity by a probabilistic analysis as follows. Assume that in a list of peers sorted according to the serving capacity for layer l, the $k+1$st peer has a $u_{p,l}$ value equal to x. Thus, the first k peers all have a $u_{p,l}$ value of $u_{p,l} \geq x$. Using this, we can obtain the expected value for the total capacity of the first k peers for layer l as:

$$E[k \times \mathbf{U}_l \mid \mathbf{U}_l > x, \mathbf{D} \geq R_l] = k \times \frac{\int_x^\infty Pr(\mathbf{U}_l = y, \mathbf{D} \geq R_l)\ y\ dy}{Pr(\mathbf{U}_l \geq x, \mathbf{D} \geq R_l)}, \quad (5)$$

where R_l is the cumulative bitrate of the first l layers, and $Pr(\mathbf{U}_l = y)$ refers to the probability density function for $\mathbf{U}_l$ at

y. The two conditions considered in this expected value capture the fact that the involved peers in this calculation already have an upload bandwidth higher than the $k+1$st peer, and a download bandwidth that can afford the first l layers; clearly, peers that are not capable of downloading the first l layers cannot be an uploader for that layer.

To obtain the expected capacity of the first k peers, we need to integrate (5) with respect to x, and consider all possible combinations of peers in an arbitrary sorted list of peers. For the candidate $k+1$st peer that we considered in (5), there are $\binom{n}{1}$ choices. Then, there are $\binom{n-1}{k}$ choices for selecting the first k peers among the remaining ones. Hence, the expected value of $f_{opt}(e,l,r_l,n)$ is calculated as:

$$f_{opt}(e,l,r_l,n) = \int_0^\infty \binom{n}{1}\binom{n-1}{k} \underbrace{Pr(\mathbf{U}_l = x \mid \mathbf{D} \geq R_l)}_{\text{I}} \times \underbrace{Pr(\mathbf{U}_l > x \mid \mathbf{D} \geq R_l)^k}_{\text{II}} \times \underbrace{Pr(\mathbf{U}_l < x \mid \mathbf{D} \geq R_l)^{n-k-1}}_{\text{III}} \times \underbrace{E[k \times \mathbf{U}_l \mid \mathbf{U}_l > x, \mathbf{D} \geq R_l]}_{\text{Eq. (5)}}\, dx, \quad (6)$$

where term I is the probability that the candidate $k+1$st peer has a capacity of x for serving layer l, term II is the probability that there are k peers with higher capacity than x, and term III is the probability that there are $n-k-1$ peers with lower such capacity. The product of these terms indicates the probability that the peer we assumed as the $k+1$st peer is actually the $k+1$st among a total of n.

We now consider the case where peers are served randomly, as opposed to employing the SCA algorithm proposed in Section III. In this case we have:

$$f_{rnd}(e,l,r_l,n) = E[k \times \mathbf{U}_l | \mathbf{D} \geq R_l] = k \times E[\mathbf{U}_l | \mathbf{D} \geq R_l] = k \times \frac{\int_0^\infty Pr(\mathbf{U}_l = x, \mathbf{D} \geq R_l)\, x\, dx}{Pr(\mathbf{D} \geq R_l)}. \quad (7)$$

Finally, we note that if the number of requesting peers is less than or equal to the number of peers that can be served, i.e., $k \geq n$, we can serve all peers that are at the segment:

$$f_{all}(e,l,r_l,n) = n \times E[\mathbf{U}_l \mid \mathbf{D} \geq R_l]. \quad (8)$$

C. Distribution of the Complete Video

We now use the formulas obtained for $f_{opt}(\cdot)$, $f_{rnd}(\cdot)$, and $f_{all}(\cdot)$ to calculate the capacity of the whole network. We first extend these functions to capture the dynamics of the network including arrivals and failures at one segment. We then employ these functions to generalize our analysis from the distribution of one segment to that of the whole video.

1) Peers Dynamics: To accommodate peer arrivals and departures (or failures) that may happen at different time instances, we define $o = \hat{f}(e,l,r_l,\mathbf{N}_t,\epsilon)$ as the output capacity for serving a segment t where the number of peers at the segment is represented by the random variable $\mathbf{N}_t$ (obtained shortly) and the failure probability for peers at the segment is ϵ. $\hat{f}(e,l,r_l,\mathbf{N}_t,\epsilon)$ is calculated as follows. Let $\mathbf{N}_{t,l}$ be a random variable representing the number of peers at segment t demanding layer l. The probability distribution of $\mathbf{N}_{t,l}$, given the number of peers $\mathbf{N}_t$, is given as:

$$\begin{aligned} &Pr(\mathbf{N}_{t,l} = n \mid \mathbf{N}_t) \\ &= \sum_{i=n}^{\infty} Pr(\text{there are } i \text{ peers}) \times Pr(n \text{ of the } i \text{ demand } l) \\ &= \sum_{i=n}^{\infty} Pr(\mathbf{N}_t = i)\binom{i}{n} Pr(\mathbf{D} \geq R_l)^n \times Pr(\mathbf{D} < R_l)^{i-n}. \end{aligned} \quad (9)$$

Accordingly, we calculate $\hat{f}(e,l,r_l,\mathbf{N}_t,\epsilon)$ as:

$$\begin{aligned} &\hat{f}(e,l,r_l,\mathbf{N}_t,\epsilon) \\ &= \sum_{n=0}^{k} Pr(\mathbf{N}_{t,l} = n) \times f_{all}(e,l,r_l,n) \times (1-\epsilon) \\ &\quad + \sum_{n=k+1}^{\infty} Pr(\mathbf{N}_{t,l} = n) \times f_{opt|rnd}(e,l,r_l,n) \times (1-\epsilon). \end{aligned} \quad (10)$$

2) Distribution of Video Segments Over Time: Consider the steady state of the network, where some peers are watching the video at segments $s = 1,\ldots,S$ and some peers have finished watching and are seeding for up to T_{seed} time units. We assume that the throughput of the network in the steady state is smaller than the demand of the peer population. Otherwise, we reduce the bandwidth or even turn off the seed server; note that the peer population is usually not capable of supporting its demand, given the highly asymmetric user bandwidths. We estimate the steady-state throughput of the network for both cases of random and using the SCA algorithm.

If serving randomly, the expected capacity of k arbitrary peers at segment t for serving layer l is $k \times E[\,\mathbf{U}_l \mid \mathbf{D} \geq \sum_{i=1}^{l} r_i(t)\,]$, as addressed in (7). If this value is less than $k \times r_l(t)$, which is the bandwidth required to serve layer l to k peers, then layer l of segment t will eventually vanish in the network since a random peer receiving it cannot serve it completely, e.g., if two peers receive the complete layer at a given time instance then only one peer can receive it at the next time instance—partial layers are of no use as they cannot be decoded. In other words, the number of peers that can be served with layer l of segment t constantly decreases and the steady-state capacity for serving that layer converges to zero. On the other hand, if the aforementioned expected value is greater or equal to $k \times r_l(t)$, eventually all peers at t that demand layer l can receive it, even if we initially seed this layer to a few random peers only. Accordingly, using (7) we can determine the video layers that survive and continue being streamed between peers, and those that do not. Using this, the video quality that is expected to be served in the network can be readily obtained, as we do in our sample analysis in Section V.

On the other hand, if peers are served according to their upload capacity, the fate of a particular layer is not necessarily either vanishing or being served to everybody. As an extreme example, if at every segment we only have 1 peer capable of serving layer l, and all other peers are incapable of that, we can still have 1 peer at each segment receiving layer l while the

others receive only up to the $l-1$st layer. We can have more peers receiving layer l by using the seed server. For instance, we can have more than 2 peers receiving the layer if we seed only one layer bitrate, since the peer we seed to will itself serve to others a portion of the layer. We now determine the number of peers that can receive layer l in the steady state as a function of the bitrate seeded.

Recall that the number of peer arrivals is represented by the random variable $\mathbf{N}$, and the failure probability at a segment t is $\gamma(t)$. Considering peers that may have left/failed at earlier segments than t, the number of peers at segment t can be represented by the random variable $\mathbf{N}_t$ as:

$$\mathbf{N}_t = \mathbf{N} \times \big(\, 1 - \sum_{i=0}^{t-1} \gamma(i) \,\big). \tag{11}$$

Once the distribution of layer l of segment t is started, at each time instance i we have:

$$x_{t,l}^{i} = \hat{f}\Big(\underbrace{x_{t,l}^{i-1} \times \frac{S + T_{seed}}{S} + C_{t,l}}_{\text{I}},\ l,\ r_l(t),\ \mathbf{N}_t,\ \gamma(t) \Big), \tag{12}$$

where $x_{t,l}^{i}$ $(i \geq 1)$ is the layer-l throughput of peers at segment t at time instance i of the system; $x_{t,l}^{0}$ is defined as zero. Term I in (12) represents the capacity served for layer l to peers at segment t, which consists of $C_{t,l}$, the capacity served by the seed server for layer l of segment t, and $x_{t,l}^{i-1}$, the capacity served by other peers (subscript v in $C_{t,l}$ and $x_{t,l}$ omitted for clarity, as $\hat{f}(\cdot)$ is applied for each video file separately). The coefficient $S + T_{seed}/S$ is to take into account the expected capacity shared by seeding peers—this coefficient has a few subtle details capturing the dynamics of the network and the probability that a peer at segment t stays for seeding, which we do not include here as they do not have a considerable effect. Note that in an actual system we are only given a total seed server capacity C, not particular $C_{t,l}$ values used in (12). The way we divide C among $C_{t,l}$ values to conduct the above calculation depends on the method used for allocation of seed servers. If the allocation is done properly (i.e., according to the population requesting different videos/pieces, as our algorithm in Section III-B does), in our analysis we can first divide C over the videos according to their viewer population, and then over segments of each video according to the $\mathbf{N}_t$ value of each segment t and the bitrate/quality of the video at that segment $(r(t), q(t))$. Finally, we divide the capacity given to each segment t over different layers based on the demand for the layers (according to $\mathbf{D}$), the bitrate/quality of the layers, and the server allocation method (random or contribution-based). Since conducting our analysis for a given network consists of several iterations over (12) for different segments and layers, we perform the above step in each iteration.

We first note that $x_{t,l}^{i}$ values converge over time (i.e., for $i = 1, 2, \ldots$), in either case of random or SCA serving. This is intuitively true, and is specifically because $x_{t,l}^{i}$ values are monotonic with i, i.e., either constantly non-increasing or constantly non-decreasing. Otherwise, for instance, we could have the case that 5 peers, either randomly selected or just the top 5 high-capacity peers, can serve 8 peers, but 8 peers (selected the same way) can serve only 7 peers, which is not possible. As $x_{t,l}^{i}$ is monotonic and it is finite, it converges to a specific value. We illustrate a sample of this convergence in Section V.

Denoting the converged value of $x_{t,l}^{i}$ simply by $x_{t,l}$, we can now determine the steady-state number of peers that are expected to receive layer l of segment t. This number, denoted by $m_{t,l}$, is obtained as:

$$m_{t,l} = \frac{x_{t,l}}{r_l(t)}. \tag{13}$$

Using the $m_{t,l}$ values we can estimate the steady-state throughput of the system along a variety of metrics. For example, the total bitrate and the average video quality served to peers can be calculated as:

$$\text{Bitrate served} = \sum_{t=1}^{S} \sum_{l=1}^{L} m_{t,l} \times r_l(t). \tag{14}$$

$$\text{Average quality} = \frac{1}{\sum_{t=1}^{S} \mathbf{N}_t} \sum_{t=1}^{S} \Big(\mathbf{N}_t \sum_{l=1}^{L} m_{t,l} \times q_l(t) \Big), \tag{15}$$

in which the use of $\mathbf{N}_t$ captures that the expected number of peers at different segments is not the same.

D. Analysis of Flash Crowd Scenarios

Using our estimate of the throughput of the system, we can analyze the capacity of the system for supporting sudden extensive loads such as a flash crowd. In a flash crowd event, a large number of peers arrive at the system at about the same time. Taking advantage of the flexibility of scalable streams, a system may sustain in these circumstances and support the new peers by temporarily lowering the video quality delivered in the network until the system recovers from the shock and returns to a stable state. Our analysis can determine the robustness of a P2P system against flash crowds, which we define as the maximum number of peers that can be admitted to the system at once. This maximum can be achieved when we serve only a minimum video quality to all peers, i.e., the base layer. Thus, having estimated the total throughput of the network, we can calculate the number of minimal substreams that can be served using this throughput, i.e., an upper bound on the maximum number of peers that can be served when a flash crowd arrives. We assume that a peer keeps the video file in its buffer until it finishes watching the video. Thus, all peers can serve the base layer of the first few segments of the video. Hence, the maximum flash crowd size N_{FC} that can be supported is calculated as:

$$N_{FC} = \frac{1}{r_1(1)} \Big(C + \sum_{t=1}^{S} \sum_{l=1}^{L} m_{t,l} \times r_l(t) - \underbrace{\sum_{t=1}^{S} \mathbf{N}_t \times r_1(t)}_{\text{I}} \Big), \tag{16}$$

where term I represents the bitrate served for delivering the base layer to peers that were already in the network. Equation (16) calculates the number of base layers that can be served to newly arrived peers. Note that it is the serving of the first few segments that is most important for sustaining a flash crowd. After the first few segments, peers arrived in the flash crowd also contribute to

uploading, and the throughput of the system increases accordingly.

E. Summary

We presented a general analytical model for studying the behavior of P2P streaming systems. We divide a video file into S segments, and analyze the bandwidth consumed and the bandwidth contributed by peers that are watching each video segment s for $s = S, S-1, \ldots, 1$. This part of the analysis is done using (10). The functions needed for calculation of (10) are derived in its preceding equations. Then, we plug the single-segment analysis function obtained in (10) into (12), which analyzes the distribution of each video segment/layer over time. Using (12) and (13), we can obtain the number of peers that are expected to receive each video segment/layer of the video stream in the steady state. These numbers are then used to predict the total bitrate that is going to be served in the system in the long term (14), the average video quality delivered to peers (15), the capability of the system for supporting flash crowds of peers (16), or others user-defined metrics for capturing the performance of the system.

V. Model Application and Analysis

In this section, we show the details of employing the proposed analytical model for analyzing a sample P2P streaming system. This is to demonstrate how our general analysis in Section IV can be used in practice. P2P streaming systems with different characteristics than considered in this section can be analyzed in a similar manner.

A. Analysis

The sample P2P streaming network we analyze is specified as follows. Suppose we distribute a video file consisting of L layers, each at a fixed rate of r kbps. The video length is S segments. We consider both random and contribution-based (SCA) serving of peers. The download bandwidth of peers is uniformly distributed in $[0, M]$ kbps, and the upload bandwidth of each peer is a constant fraction α of its download bandwidth, which makes it uniformly distributed in $[0, \alpha M]$ kbps. Peers arrive according to a Poisson distribution with an average arrival rate of λ, and stay after they finish watching the video for seeding for up to T_{seed} time units. A fraction of peers leave the system before watching the complete video and seeding. These departures happen with probability $\hat{\gamma}_t$ for peers at segment t $(1 \le t \le S)$.

We customize the model and equations derived in the previous section for the network described above. First, we need to calculate the functions $f_{all}(\cdot)$, $f_{rnd}(\cdot)$, and $f_{opt}(\cdot)$, which take as input the bandwidth e kbps for serving the r-kbps layer l to n peers. For this purpose, we need to obtain the closed form expression for $E[\mathbf{U}_l \mid \mathbf{D} \ge R_l]$, the expected value of the upload capacity for layer l by a peer that can download layer l, since this value is repeatedly used in the following equations. This is done using the distribution of bandwidths as:

$$E[\mathbf{U}_l \mid \mathbf{D} \ge R_l] = E[\mathbf{U}_l \mid \mathbf{U} \ge \alpha r l],$$

where $E[\mathbf{U}_l \mid \mathbf{U} \ge z]$ for any arbitrary z is calculated according to the way a peer shares its upload bandwidth among layers (4) and the distribution of the upload bandwidth of peers, which is uniform in our case. More specifically, denoting by $U_l(u)$ the layer-l upload bandwidth of a peer whose total upload bandwidth is u, according to (4) we have:

$$U_l(u) = \begin{cases} \frac{u}{L} & rL \le u \le \alpha M \\ r & rl \le u < rL \\ u - r(l-1) & r(l-1) \le u < rl \\ 0 & 0 \le u < r(l-1), \end{cases}$$

which leads to calculation of the desired expected value $E[\mathbf{U}_l \mid \mathbf{U} \ge z]$ as in (17). The first row in (17) corresponds to the case where the upload bandwidth of a peer is higher than the total bitrate of video layers, in which a bandwidth u is evenly divided among the L layers. The second row adds to this case the possibility that the upload bandwidth is less than the bitrate of all layers, but higher than rl, the bitrate of the first l layers. In this case, the bitrate at which layer l is streamed to other peers is just equal to r. The third row adds the case that the peer's upload bandwidth is less than the bitrate of the first l layers, but higher than the first $l-1$ of them. In this case, it can upload layer l at a rate less than r. Finally, the last row considers the case where the peer cannot upload layer l at all.

$$E[\mathbf{U}_l \mid \mathbf{U} \ge z] = \frac{\int_z^{\infty} Pr(\mathbf{U} = u) U_l(u)\, du}{Pr(\mathbf{U} \ge z)} = \frac{\int_z^{\alpha M} U_l(u)\, du}{\alpha M - z}$$
$$= \begin{cases} \frac{1}{\alpha M - z} \int_z^{\alpha M} \frac{u}{L}\, du & rL \le z \le \alpha M \\ \frac{1}{\alpha M - z} \int_z^{rL} r\, du & \\ \quad + E[\mathbf{U}_l \mid \mathbf{U} \ge rL] & rl \le z < rL \\ \frac{1}{\alpha M - z} \int_z^{rl} (u - r(l-1))\, du & \\ \quad + E[\mathbf{U}_l \mid \mathbf{U} \ge rl] & r(l-1) \le z < rl \\ \frac{1}{\alpha M - z} \int_z^{r(l-1)} 0\, du & \\ \quad + E[\mathbf{U}_l \mid \mathbf{U} \ge r(l-1)] & 0 \le z < r(l-1). \end{cases} \quad (17)$$

Now that we obtained $E[\mathbf{U}_l \mid \mathbf{U} \ge z]$ for any arbitrary z, we can calculate functions $f_{all}(\cdot)$ and $f_{rnd}(\cdot)$ using (7) and (8):

$$f_{all}(e, l, r, n) = n \times E[\mathbf{U}_l \mid \mathbf{U} \ge \alpha r l]$$
$$f_{rnd}(e, l, r, n) = \lfloor \frac{e}{r} \rfloor \times E[\mathbf{U}_l \mid \mathbf{U} \ge \alpha r l].$$

Next, we calculate $f_{opt}(\cdot)$ using (6):

$$f_{opt}(e, l, r_l, n) = \int_0^{\alpha M} Pr(\mathbf{U} = u \mid \mathbf{U} \ge \alpha r l) \times Pr(\mathbf{U} \ge u \mid \mathbf{U} \ge \alpha r l)^k \times Pr(\mathbf{U} < u \mid \mathbf{U} \ge \alpha r l)^{n-k-1} \times E[\mathbf{U}_l \mid \mathbf{U} \ge \alpha r l, \mathbf{U} \ge u]\, du, \quad (18)$$

in which all probabilities can be easily derived according to the uniform distribution of $\mathbf{U}$ in $[0, \alpha M]$.

The next function we need to calculate is $Pr(\mathbf{N}_l = n \mid \epsilon)$, the probability distribution of the number of peers at a segment

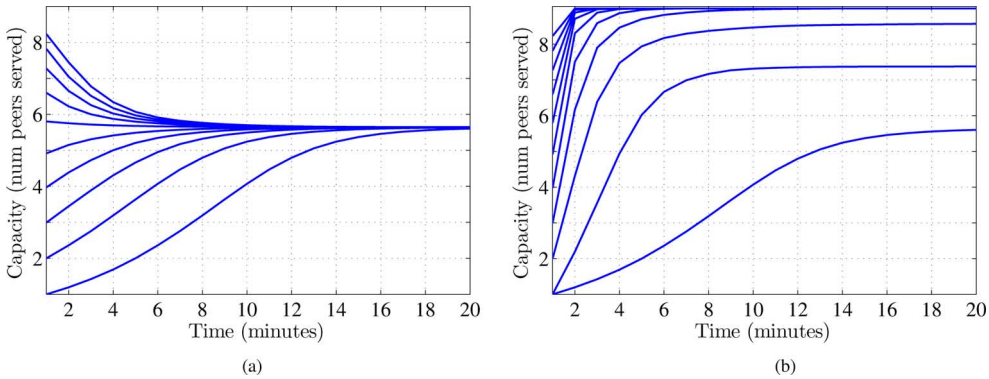


Fig. 3. The streaming capacity of peers at one segment for one layer. (a) Streaming capacity when we only seed at the first time step. The curves from bottom to top correspond to initial seed values from 1 to 10 layer rates. (b) Streaming capacity when seeding continues. The top 10 of the 11 curves correspond to seeding capacities 1 to 10 layer rates. The bottom curve is the special case of seeding with 1 layer rate at the first time step only.

which are demanding layer l, when the failure probability for peers at that segment is ϵ. This is done using (9):

$$Pr(\mathbf{N}_l = n \mid \epsilon) = \sum_{i=n}^{\infty} f_{\text{Poisson}}(\lceil \frac{i}{1-\epsilon} \rceil; \lambda) \binom{i}{n} \times (\frac{\alpha M - rl}{\alpha M})^n (\frac{rl}{\alpha M})^{i-n}, \quad (19)$$

where $f_{\text{Poisson}}(\lceil i/1-\epsilon \rceil; \lambda)$ is the probability of exactly $\lceil i/1-\epsilon \rceil$ arrivals given a Poisson distribution with parameter λ.

We can now analyze the distribution of each video segment through (10) and (12), since we have already obtained all its preliminaries in the above equations. Thus, we have:

$$x^i_{t,l} = \hat{f}(\, x^{i-1}_{t,l} \times \frac{S + T_{seed}}{S} + C_{t,l}, l, r, \underbrace{\lambda(1 - \sum_{i=1}^{t-1} \hat{\gamma}_i)}_{\text{I}}, \hat{\gamma}_t\,), \quad (20)$$

where term I denotes the expected number of peers (with Poisson distribution) at segment t, i.e., peers whose expected number was λ at the beginning of their joining, and left the system with probability $\hat{\gamma}_i$ at each time instance i. Denoting the value converged over time by $x_{t,l}$, the steady-state number of peers receiving layer l of segment t, i.e., $m_{t,l}$, is calculated using (13).

B. Numerical Results (Answering Q_2 and Q_3)

Since each video segment is analyzed individually, let us first focus on the distribution of one segment. We would like to see how many peers at a given segment can be served with a particular layer l in the long term. We then use these results as a basis for analyzing the distribution of the complete video. To carry out a numerical analysis, we fix the value of some of the parameters as follows. We suppose the distribution of a 1-hour video which we divide into 60 1-minute segments. The video has $L = 10$ layers, each at rate $r = 200$ kbps. The maximum download bandwidth of a peer is $M = 10$ Mbps and its upload bandwidth is $\alpha = 20\%$ of its download bandwidth. Peers arrive at a rate of $\lambda = 10$ peers per minute with a Poisson distribution, and after watching the video, they seed for up to $T_{seed} = 30$ minutes. 25% of peers leave the system before they finish watching and seeding, which happens with an exponential probability distribution with most of the departures taking place in the first few minutes of watching.

1) Step 1: Distribution of One Video Segment: We first note that the number of peers served with layer l of a given segment t converges over time, as discussed in Section IV-C. This can be observed in Fig. 3(a), which shows a sample result. In this figure, considering a window of peers at one segment over time, the number of peers that are served with layer 5 in each time step is plotted. Each curve, from bottom to top, corresponds to an initial seed capacity, starting from 1 layer bitrate and increasing to 10. The failure probability for peers at the segment is assumed 20%. In Fig. 3(a), independent of the initial seed, the number of peers that the network can serve with layer 5 of the segment converges to 5.6 on average, though it takes more time to boost the capacity when the initial seed is small. If we do not stop seeding after the first time step and keep seeding over time, indeed a higher bitrate is served to the peers. This increase in the throughput for streaming layer 5 of the segment is illustrated in Fig. 3(b). In this figure, the increase in the number of served peers when the seeding capacity is increased is more significant for smaller capacity values, which is particularly observable as the gap between the three lowest curves. This is because in those cases, the entire seeded bitrate is consumed by peers, and since peers will also contribute to serving, the increase in the throughput is more than the seeded bitrate. On the other hand, if the seed is more than the demand of the peers, only part of it is consumed, and the number of peers served will converge to the number demanding the layer. This is observed in Fig. 3(b) for seeding capacities of more than 3 layer bitrate. In particular, 7 out of the 11 curves gather on top of the figure. Though Fig. 3(b) corresponds to the serving of layer 5 only, serving of other layers demonstrates similar behaviors. For lower layers (1 to 4), the most bottom curve converges to higher values than 5.6, because the total upload rate of lower layers by the peers is higher. Moreover, more curves gather on top of the figure. For higher layers (6 to 10), the reverse of these behaviors is observed.

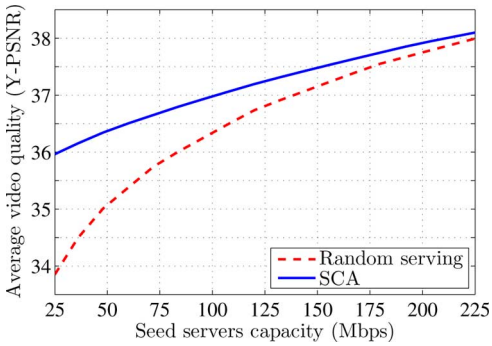


Fig. 4. Average video quality delivered to peers.

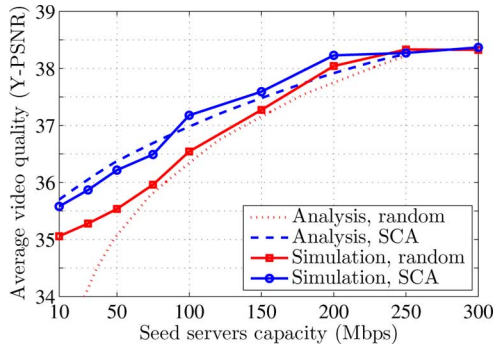


Fig. 5. Validation of the analytical model.

In summary, using our analysis and given the seeding capacity, we can determine for each layer the value to which the number of served peers at a segment converges, i.e., the $m_{t,l}$ value introduced in Section IV-C, which is the steady-state number of peers served with layer l of segment t. We use these values for analyzing the distribution of the complete video in the next step.

2) Step 2: Deriving the Throughput for the Complete Video: Using the previous step, we obtain $m_{t,l}$ for all valid t, l values. Consequently, we can calculate the throughput of the network in terms of various metrics. For example, we estimate the average video quality delivered to peers using (15) for different values of seeding capacity C. This is plotted in Fig. 4 for both cases of SCA and random serving. **Fig. 4 answers questions $\mathbf{Q}_2$ and $\mathbf{Q}_3$ discussed in Section I**: it estimates the throughput of the system in terms of video quality, and it computes the capacity needed for providing a desired level of video quality. For example, if we provide 25 Mbps seeding capacity, the average video quality delivered to peers will be 35.9 dB. If we increase this capacity to 125 Mbps, the average quality will increase to 37.3 dB. This increase continues as we provider higher capacities, and it eventually approaches a value of approximately 38.5 dB, which is the maximum quality demand of peers—note that although the video stream can provide a quality of up to 40 dB, not all peers are demanding this quality level.

VI. Evaluation and Validation of the Capacity Planning Model

In this section, we first validate our analytical model by comparing its results to those obtained from simulation. Then, we show how we can use the analysis for studying different performance aspects of P2P streaming systems.

A. Validation Using Simulation

The simulation setup is the same as mentioned in Section V-B. According to the arrival and failure/departure rates, the video length, and the peer seeding time, the number of peers in the network varies with time, but on average there are 500–600 at any time. Each simulation runs for 5 hours of simulation time. We make peers disobey our assumption about the way they share their upload bandwidth among different layers in (4) by having each $u_{p,l}$ value deviate by up to 50% from its supposed value. This is used to capture peers and network dynamics in real systems.

We run the simulation for different values of seed server capacity, and compare the results with those of our analysis. This comparison is illustrated in Fig. 5, which shows the average video quality in the steady state obtained by the analytical and the simulation study for different values of seeding capacity. Fig. 5 shows that estimations of the analysis are reasonably close to the results obtained by simulations, except for small seeding capacities. The deviation observed for these values in the case of random serving is because the analytical model assumes a certain value as the upload bandwidth of a peer for a particular layer in (4). This assumption becomes far from reality when the seeding capacity is small and it is not carefully allocated, because in this case only a few video layers are exchanged between peers and the upload bandwidths of peers for higher layers become significantly smaller than expected; and those for lower layers are larger than expected. Except for this particular case, the simulation explained in this section with dynamic P2P streaming systems confirms the accuracy of our analysis.

B. Flash Crowd Scenarios (Answering Q_4)

Using the flash crowd analysis in Section IV-D, we obtain the maximum number of peers (arriving together) that can be served by the system for different values of seeding capacity. **The results of this experiment are plotted in Fig. 6(a), which answer question Q_4 discussed in Section I**. In this figure, curves labeled "Analysis" show the upper bound on the flash crowd size that can be supported by the system, using the total serving capacity of peers and seed servers. Curves labeled "Simulation" depict the number of peers supported in a flash crowd in a simulated P2P system. To obtain these numbers, we make a set of 10 000 peers arrive in the system at the same time (at $t = 2$ hours). Shortly after the arrivals, we measure the number of peers out of these 10 000 that could receive a video stream (the base layer). Note that the analytical results for the flash crowd test only provide an upper bound on the number of peers that can be served, i.e., ideally if the peers' upload capacities could be properly utilized for each other even right after the sudden arrival of thousands of peers. Accordingly, there is a gap between the analytical upper bound and the experimental results (with a single-machine simulator) for this test. Moreover, although with a smaller domain, the results of simulation also do not exhibit a

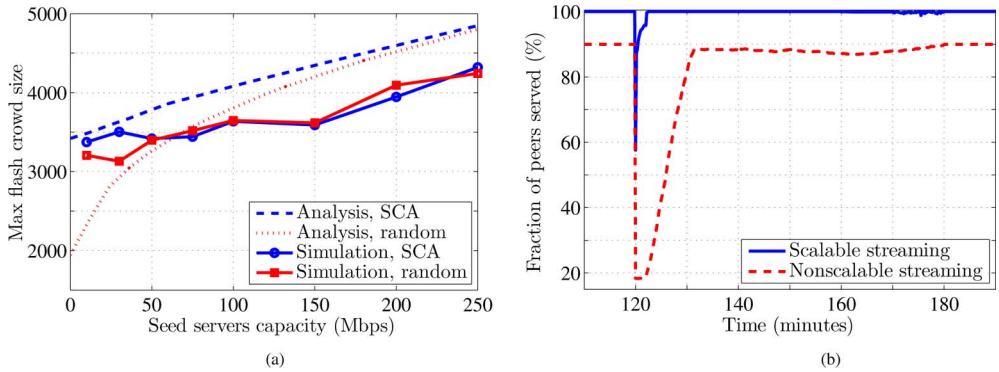


Fig. 6. Analyzing flash crowd scenarios. (a) Number of peers served in flash crowd. (b) Robustness of scalable streaming against flash crowds.

smooth behavior. This can be attributed to the high degree of dynamics introduced to the simulated systems by the large crowd of peers arriving at the same time.

Nevertheless, Fig. 6(a) shows that for smaller seed server capacities, the two serving methods can make a significant difference in terms of the flash crowd size that can be supported. For example, to support 3500 peers, approximately 70 Mbps capacity is needed for seed server in case of random serving, whereas if the serving has been done carefully until the flash crowd arrives, the seeding capacity can potentially be as low as 10 Mbps.

C. Benefits of Scalable Video Streams (Answering Q_5)

When scalable video streams are served, flash crowd cases can be handled more effectively compared to the case with nonscalable streams. This is because scalable streams can be adapted to lower bitrates for a short period, until the system recovers from the shock. With nonscalable streams, the video bitrate cannot be adapted, and many peers will have to be rejected. We analyze a flash crowd scenario in our simulator in order to derive more insights on the performance of P2P streaming systems with scalable videos as well as those with nonscalable videos, specially their robustness against flash crowd events.

We run the test for a seeding capacity of 50 Mbps. According to Fig. 6(a), 3800 is an upper bound on the size of a flash crowd that can be supported. Thus, we make a set of 3500 peers arrive to the simulated system at time instance $t = 2$ hours; the time that the system reaches a steady state is $t = 90$ min. We assume that after the crowd event happens, peers that arrive in this crowd are not able to immediately start uploading data. In particular, in our simulations those peers cannot upload any data for the first two minutes. Then, they gradually become able to do so with an expected time of five minutes. This is because in a real system, the tracker needs some time to catch up with the large amount of changes in the system and to introduce these peers as potential senders to other peers.

We consider a similar streaming scenario with nonscalable video. The bitrate of the nonscalable vide stream determines a tradeoff between the number of peers that can benefit from the system and the video quality delivered to users. Assuming that we would like to be able to support 90% of peers, we set the video bitrate to 1 Mbps. We also assume that the overhead of scalable coding is 10% [20], meaning that our nonscalable stream can provide a visual quality equal to the one provided by the corresponding scalable stream at 1.10 Mbps (35.2 dB).

Note that in case of serving nonscalable video streams, we would not have the possibility to switch to lower bitrate videos, and accordingly, many peers would have been rejected from the system. Therefore, the average video quality does not drop as it does with scalable streaming, but the system rejects many peers. To analyze this issue, we plot in Fig. 6(b) the fraction of peers who are receiving a video stream, i.e., at least the base layer. **This figure answers question Q_5 discussed in Section I**. Once the flash crowd arrives, the fraction of peers receiving a video stream drops momentarily for the scalable streaming case to a value of approximately 60%, and increases back to over 90% in less than two minutes. However, this value drops dramatically in case of nonscalable streaming. It takes more than 10 minutes for the newly arrived peers to catch up with the stream.

VII. Conclusions

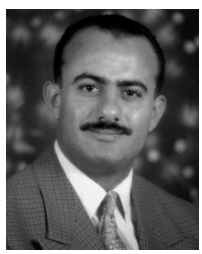
We have studied the problem of managing the seed server capacity in P2P streaming systems with scalable videos. We first formulated the problem of allocating the resources of seed servers to peers' requests for different video layers. We proved the NP-completeness of this problem, and proposed an approximation algorithm to solve it. Using our allocation algorithm, we developed an analytical model to analyze the performance of P2P streaming systems with scalable videos. The model computes the long-term throughput of the network in terms of the delivered video quality and the total served bitrate. Our analysis can help administrators of P2P streaming systems to assess the benefits of deploying a given amount of seeding resources, and to determine the cost-benefit tradeoff for provisioning a higher seeding capacity to sustain a desired level of video quality. Moreover, the proposed analysis can estimate an upper bound on the capability of a P2P system for supporting flash crowds, which is the maximum number of peers that can be admitted to the system within a short period of time. We have validated our analytical model using extensive simulations with realistic parameters of dynamic P2P streaming systems.

References

[1] PPLive. [Online]. Available: http://www.pplive.com/en/index.html.
[2] SopCast. [Online]. Available: http://www.sopcast.org/.

[3] S. Annapureddy, S. Guha, C. Gkantsidis, D. Gunawardena, and P. Rodriguez, "Exploring VoD in P2P swarming systems," in *Proc. IEEE INFOCOM'07*, Anchorage, AK, May 2007, pp. 2571–2575.
[4] Y. Cui and K. Nahrstedt, "Layered peer-to-peer streaming," in *Proc. ACM Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'03)*, Monterey, CA, Jun. 2003, pp. 162–171.
[5] M. Hefeeda and C. Hsu, "Rate-distortion optimized streaming of fine-grained scalable video sequences," *ACM Trans. Multimedia Comput., Commun., Appl. (TOMCCAP)*, vol. 4, no. 1, pp. 2:1–2:28, Jan. 2008.
[6] X. Hei, C. Liang, J. Liang, Y. Liu, and K. Ross, "A measurement study of a large-scale P2P IPTV system," *IEEE Trans. Multimedia*, vol. 9, no. 8, pp. 1672–1687, Dec. 2007.
[7] O. Hillestad, A. Perkis, V. Genc, S. Murphy, and J. Murphy, "Adaptive H.264/MPEG-4 SVC video over IEEE 802.16 broadband wireless networks," in *Proc. Packet Video Workshop (PV'07)*, Lausanne, Switzerland, Nov. 2007, pp. 26–35.
[8] H. Hu, Y. Guo, and Y. Liu, "Mesh-based peer-to-peer layered video streaming with taxation," in *Proc. ACM Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'10)*, Amsterdam, The Netherlands, Jun. 2010, pp. 27–32.
[9] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. New York: Springer, 2004.
[10] R. Kumar, L. Yong, and K. Ross, "Stochastic fluid theory for P2P streaming systems," in *Proc. IEEE INFOCOM'07*, Anchorage, AK, May 2007, pp. 919–927.
[11] X. Lan, N. Zheng, J. Xue, X. Wu, and B. Gao, "A peer-to-peer architecture for efficient live scalable media streaming on Internet," in *Proc. ACM Multimedia Conf.*, Augsburg, Germany, Sep. 2007, pp. 783–786.
[12] B. Li and J. Liu, "Multirate video multicast over the Internet: An overview," *IEEE Network*, vol. 17, no. 1, pp. 24–29, Feb. 2003.
[13] S. Liu, R. Zhang-Shen, W. Jiang, J. Rexford, and M. Chiang, "Performance bounds for peer-assisted live streaming," in *Proc. ACM Conf. Measurement and Modeling of Computer Systems (SIGMETRICS'08)*, Annapolis, MD, Jun. 2008, pp. 313–324.
[14] K. Mokhtarian, "Efficient and secure delivery of scalable video streams," Master's thesis, Simon Fraser Univ., Surrey, BC, Canada, 2009.
[15] K. Mokhtarian and M. Hefeeda, "Efficient allocation of seed servers in peer-to-peer streaming systems with scalable videos," in *Proc. IEEE Int. Workshop on Quality of Service (IWQoS'09)*, Charleston, SC, Jul. 2009, pp. 1–9.
[16] K. Mokhtarian and M. Hefeeda, "Analysis of peer-assisted video-on-demand systems with scalable video streams," in *Proc. ACM Multimedia Systems Conf. (MMSys'10)*, Scottsdale, AZ, Feb. 2010, pp. 133–144.
[17] N. Parvez, C. Williamson, A. Mahanti, and N. Carlsson, "Analysis of Bittorrent-like protocols for on-demand stored media streaming," in *Proc. ACM Conf. Measurement and Modeling of Computer Systems (SIGMETRICS'08)*, Annapolis, MD, Jun. 2008, pp. 301–312.
[18] R. Rejaie and A. Ortega, "PALS: Peer-to-peer adaptive layered streaming," in *Proc. ACM Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'03)*, Monterey, CA, Jun. 2003, pp. 153–161.
[19] T. Schierl, T. Stockhammer, and T. Wiegand, "Mobile video transmission using scalable video coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 9, pp. 1204–1217, Sep. 2007.
[20] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the scalable video coding extension of the H.264/AVC standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 9, pp. 1103–1120, Sep. 2007.
[21] T. Small, B. Liang, and B. Li, "Scaling laws and tradeoffs in peer-to-peer live multimedia streaming," in *Proc. ACM Multimedia Conf.*, Santa Barbara, CA, Oct. 2006, pp. 539–548.
[22] Y. Tu, J. Sun, M. Hefeeda, Y. Xia, and S. Prabhakar, "An analytical study of peer-to-peer media streaming systems," *ACM Trans. Multimedia Comput., Commun., Appl.*, vol. 1, no. 4, pp. 354–376, Nov. 2005.
[23] X. Xiao, Y. Shi, and Y. Gao, "On optimal scheduling for layered video streaming in heterogeneous peer-to-peer networks," in *Proc. ACM Multimedia Conf.*, Vancouver, BC, Canada, Oct. 2008, pp. 785–788.
[24] D. Xu, M. Hefeeda, S. Hambrusch, and B. Bhargava, "On peer-to-peer media streaming," in *Proc. Int. Conf. Distributed Computing Systems (ICDCS'02)*, Vienna, Austria, Jul. 2002, pp. 363–371.
[25] D. Xu, S. Kulkarni, C. Rosenberg, and H. Chai, "Analysis of a CDNP2P hybrid architecture for cost-effective streaming media distribution," *Multimedia Syst.*, vol. 11, no. 4, pp. 383–399, Mar. 2006.
[26] H. Yin, X. Liu, T. Zhan, V. Sekar, F. Qiu, C. Lin, H. Zhang, and B. Li, "Design and deployment of a hybrid CDN-P2P system for live video streaming: Experiences with LiveSky," in *Proc. ACM Multimedia Conf.*, Beijing, China, Oct. 2009, pp. 25–34.
[27] X. Zhang, J. Liu, B. Li, and T. Yum, "CoolStreaming/DONet: A data-driven overlay network for peer-to-peer live media streaming," in *Proc. IEEE INFOCOM'05*, Miami, FL, Mar. 2005, vol. 3, pp. 2102–2111.
[28] Y. Zhou, D. Chiu, and J. Lui, "A simple model for analyzing P2P streaming," in *Proc. IEEE Int. Conf. Network Protocols (ICNP'07)*, Beijing, China, Oct. 2007, pp. 226–235.

Kianoosh Mokhtarian (S'09) received the B.Sc. degree in Software Engineering from Sharif University of Technology, Tehran, Iran, in 2007 and the M.Sc. degree in Computing Science from Simon Fraser University, Surrey, BC, Canada, in 2009. He is currently a Ph.D. student at the University of Toronto, ON, Canada. His research interests include distributed systems, peer-to-peer networks and multimedia networking.



Mohamed Hefeeda (S'01–M'04–SM'09) received the B.Sc. and M.Sc. degrees from Mansoura University, Egypt in 1994 and 1997, respectively, and the Ph.D. degree from Purdue University, West Lafayette, IN, in 2004. He is an associate professor in the School of Computing Science, Simon Fraser University, Surrey, BC, Canada, where he leads the Network Systems Lab. His research interests include multimedia networking over wired and wireless networks, peer-to-peer systems, and cloud computing. In 2011, he was awarded one of the prestigious NSERC Discovery Accelerator Supplements (DAS). His paper on the hardness of optimally broadcasting multiple video streams with different bitrates won the Best Paper Award in the IEEE Innovations 2008 conference. In addition to publications, he and his students develop actual systems, such as PROMISE, pCache, svcAuth, pCDN, and mobile TV testbed, and contribute the source code to the research community. The mobile TV testbed software developed by his group won the Best Technical Demo Award in the ACM Multimedia 2008 conference. He serves as the Preservation Editor of the ACM Special Interest Group on Multimedia (SIGMM) Web Magazine. He has served as the program chair of the ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2010). In addition, he has served on many technical program committees of major conferences in his research areas, including ACM Multimedia, ACM Multimedia Systems, ACM/SPIE Multimedia Computing and Networking (MMCN), IEEE Conference on Network Protocols (ICNP), and IEEE Conference on Communications (ICC). He serves on the editorial boards of the ACM Transactions on Multimedia Computing, Communications and Applications (ACM TOMCCAP), and the Journal of Multimedia.