Push-to-Peer Video-on-Demand system: design and evaluation

Kyoungwon Suh[†], Christophe Diot[‡], Jim Kurose[†]
Laurent Massoulie[‡], Christoph Neumann[‡], Don Towsley[‡]
Matteo Varvello[*]

[†]University of Massachusetts, Amherst, MA 01003, US
[‡]Thomson Research Lab, Paris, France
[*]Eurecom, Sophia-Antipolis, France

UMass Computer Science Techincal Report 2006-59
Nov-5-2006

ABSTRACT

We propose Push-to-Peer, a peer-to-peer approach to cooperatively stream video. The main departure from previous work is that content is proactively pushed to peers, and persistently stored before the actual peer-to-peer transfers. The initial content placement increases content availability and improves use of peer uplink bandwidth.

Our specific contributions are: (i) content placement and associated pull policies that allow optimal use of uplink bandwidth and perfect balancing of download rates among competing downloads; (ii) performance analysis of such policies in the case of controlled environments, such as DSL networks under ISP control; (iii) distributed load balancing strategies for initial selection of serving peers; (iv) distributed strategies to cope with dynamic uplink bandwidth. The latter two contributions allow to handle the case of uncontrolled environments, such as the public Internet.

1. INTRODUCTION

Over the past five years, there has been considerable research in the use of peer-to-peer networks for distributing both live [6, 26, 20, 19] and stored [7, 2] video. In such systems, peer interest plays the central role in content transmission and storage - a peer *pulls* content only if the content is of interest. Once pulled content has been stored locally, the peer may then in turn distribute this content to yet other self-interested peers. Such a pull-based system design is natural when the individual peers are autonomous and self-interested. However, when the individual peers are under common control, for example in the case of a residential home gateway or set-top box under the control of a network or content provider, a richer range of system designs becomes possible.

In this paper, we investigate the design space of a *Push-to-Peer* Video-on-Demand (VoD) system. In such a system, video is first pushed (e.g., from a content creator) to a population of peers. This first step is performed under provider or content-owner control, and can be performed during times of low network utilization (e.g., early morning). Note that as a result of this push phase, a peer may store content in which it itself has no interest, unlike traditional pull-only peer-to-peer systems. Following the push phase, peers seeking specific content will then pull content of interest from other peers, as in a traditional peer-to-peer system. The Push-to-Peer approach is well-suited to cooperative distribution of stored video among set-top boxes in DSL networks, where the set-top boxes themselves operate under provider control. We believe, however, that the Push-to-Peer approach is more generally applicable to cases in which peers are long-lived, and willing to have content proactively pushed to them before video distribution among the cooperating peers begins.

In this paper, we consider the design and analysis of a Push-to-Peer system in a network of long-lived peers in which upstream bandwidth and peer storage are the primary resources constraints. We begin by considering a *controlled* environment, with a set of always-on peers, constant available bandwidth among the peers, and the possibility of centralized control, assumptions appropriate in the specific setting of a VoD system consisting of set-top boxes within a single DSLAM [11] in a DSL network. We begin by describing an idealized policy for placing video data at the peers during the push phase - full striping - and its consequent pull policy for downloading video. We demonstrate that no other data placement policy can satisfy a higher demand rate without blocking (which occurs when a peer is unable to download a video at a sufficiently high rate to support playback). We also consider the practical case in which the number of peers from which a peer can download is bounded, and propose two policies - constrained striping and a coding scheme - for handling this constraint. We analyze the performance of these policies (in terms of blocking under a no-wait blocking model, and delay under a model in which blocked requests are queued until they can be served). Our performance models can be used not only to quantitatively analyze system performance but also to dimension systems so that a given level of user performance is realized - an important consideration if Push-to-Peer is provided as a billable service by the network provider. We also consider the case of prefix caching at the peers.

While the discussion above has focused on controlled en-

vironments, we are also interested in uncontrolled environments, e.g., as in the public Internet. Our initial work here analyzes the performance of a distributed policy for assigning equal number of peers to each box (in the absence of centralized control) and analyzes the properties of two different approaches for handling the case of time-varying upload capacities. We believe this paper thus provides a foundation for both a practical and a theoretical understanding of the design and performance of this new class of peer-to-peer based VoD systems.

The remainder of this paper is structured as follows. In Section 2, we describe the controlled DSLAM setting, and the push and pull phases in more detail. We also summarize some of the important differences between the Push-to-Peer and traditional peer-to-peer approaches for VoD. In Section 3, we describe three policies for placing video data at the peers during the push phase. In Section 4 we analyze the performance of the previous schemes under both a blocked-calls-lost and blocked-calls-queued model. We apply those analytical results to address the problem of prefix sizing. In Section 5 we turn our attention to less controlled environments, and investigate the performance load balancing policies and the properties of two approaches for time-varying upload capacities. Section 6 presents a discussion of the related work. Section 7 concludes this paper.

2. NETWORK SETTING AND PUSH-TO-PEER OPERATION

In this section, we describe the network setting for the Push-to-Peer architecture and overview of push and pull phases of operation. We also describe our model of video playback, in terms of user requirements and performance metrics.

We will describe the Push-to-Peer architecture in the context of a number of always-on set-top boxes (STBs) or Residential Home Gateways (RHGs) that collectively sit below a DSLAM in a DSL network and cooperatively distribute video amongst themselves, as described below. We focus on this particular controlled network setting because it provides a concrete real-world scenario, but stress that the the Push-to-Peer approach is more generally applicable to cases in which peers are long-lived, and willing to have content proactively pushed to them before video distribution among the cooperating peers begins.

Figure 1 illustrates the network setting. The Push-to-Peer system is composed of a content server, a control server, and boxes at the user premises. The content server, located in content provider's premises, pushes content to the boxes during the *push* phase, as described below. A control server is also located in content provider's premise; it provides a directory service to boxes in addition to management and control functionalities. The always-on STBs or RHGs reside at the customer premises. Although there are important technological and commercial differences between STBs and RHGs, we will refer to these devices generically as *boxes* in the remainder of this paper, since their crucial capabilities - the ability to download, upload, and store video under provider control - are common to both STBs and RHGs.

Content distribution proceeds in two phases in our Push-to-Peer system.

- **Push Phase.** During the push phase, shown in Figure 1, the content server pushes content to each of the boxes. We envision this happening periodically, when bandwidth is plentiful (e.g., in the early AM hours). After pushing content to the peers, the content server then disconnects (i.e., does not provide additional content push), until the next push phase. A crucial issue for the push phase is that of data placement: what portions of which videos should be placed on which boxes; we address this problem in Section 3.

- **Pull Phase.** In the pull phase, shown in Figure 2, boxes respond to user commands to play content. Since a box will typically not have all of the needed content at the end of the push phase, it will need to retrieve missing content from its peers. While it is possible for the boxes to proactively push content among themselves (not in response to user commands) we do not consider that possibility here.

We make the following assumptions about the DSL network, and the Boxes at the user premises:

- **Upstream and downstream bandwidth.** We assume that the upstream bandwidth from the boxes to the DSLAM is a constrained resource, and is smaller than the video encoding/playback rate. We will consider the cases that the available upstream bandwidth for the Push-to-Peer system is either fixed, or can vary over time. We assume that if a peer is uploading video to N different peers, then the upstream bandwidth is equally shared among those N peers. We also assume that video is transferred reliably, either through FEC or some ARQ mechanism. We assume that the downstream bandwidth is large enough so that it is never the bottleneck when a peer is downloading video from other (possibly many other) peers (instead, the upstream bandwidths at those other peers are collectively the limiting resource). We thus also assume that the downstream bandwidth is larger than the video encoding/playback rate.

- **Peer storage.** We assume that boxes have hard-disks that can store content that is pushed to the box during the push phase. This content can then be uploaded to other peers upon request, during the pull phase. The disk may also store movie prefixes, that are used locally at the STB to decrease startup delay, as discussed in Section 4. We note that when an STB needs to pull video from other boxes for movie playout, this video must also be stored in a local playout buffer, but we do not consider the (relatively small) requirements of this playout buffer here.

- **Peer homogeneity.** We assume homogeneous peers, i.e., that all peers have the same upstream link bandwidth and the same amount of hard disk storage.

The requests for video playout are generated by users located at the boxes. We assume that a user views a video from start to finish, with no VCR actions. A box will start to play a video as early as possible (without waiting for the whole movie being available in its memory), as long as it is guaranteed to be able to play a video without buffer under-run. Specifically, suppose that at time 0, a box starts to download the missing portion of movie in parallel from

other boxes. At time t, the playback starts if it is guaranteed that there is a schedule of available boxes with the needed missing content and sufficient upstream bandwidth to supply each missing block before its playout time.

We define a minimum fetch size of a partial copy of movie for parallel downloading to support pseudo-streaming capability. Specifically, we divide a movie of size L into chunks of equal size W, which we call windows. A window is the minimum unit of segment of a movie which should be available in the local memory of the viewer, in order to be played back. Therefore, the startup delay will be represented as the lead time to play a set of initial x windows (mostly $x = 1$). To enable parallel downloading, a given window is divided into smaller blocks of equal size and those blocks are distributed to the boxes. In the block distribution schemes we consider, typically each box will only keep a fraction of the blocks that comprise a window.

Because we are interested in systems in which the startup delay is relatively small, we will require that the aggregate rate at which a peer receives downloaded video from the other peers to be at least as large as the video encoding/playout rate [1]. If a user requests video playout and this aggregated upload rate is not available, the video playout request is *blocked*. We will mostly consider the case where blocked calls are lost, but will also consider the case where blocked calls can be queued.

Since the Push-to-Peer provides on-demand *video* streaming, we consider the maximum start-up delay and request blocking rate as the metrics to measure the performance of the Push-to-Peer system.

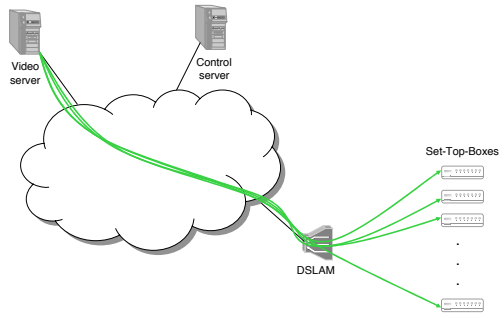

Figure 1: Push phase

The main notations that will be used throughout the paper are listed in Table 1.

3. DATA PLACEMENT AND PULL POLICIES

In this section we first propose the full-striping data placement scheme. Next we establish its optimality in terms of the demand rates it can accommodate. We then introduce the so-called constrained striping and coding data placement schemes. In contrast to full striping, these allow a box to download a video from a small number of boxes. This is useful when the number of simultaneous connections that a box can support is constrained. We then state an optimality property of the coding scheme, in terms of the demand rates it can accommodate.

[1]We treat the data blocks already pushed to a peer's memory as if they are also downloaded from other peers within 0 time unit, when the aggregate downloading rate is computed.

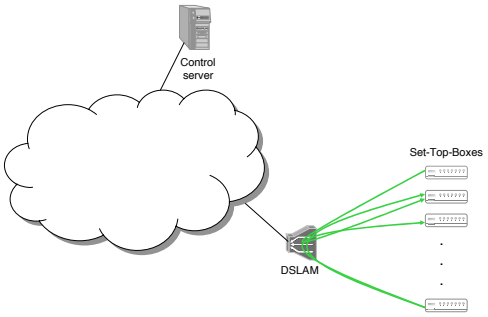

Figure 2: Pull phase

3.1 Full-Striping scheme

A full-striping scheme stripes each window of a movie to all M boxes. Specifically, every window of size W is divided into M blocks of size W/M and each block is pushed to only one box *proactively*. Consequently, each box keeps a *distinct* block of a window. A full copy of a given window is reconstructed by downloading $(M-1)$ distinct blocks for the window in parallel from $(M-1)$ boxes *on demand*[2].

A box serves admitted requests according to the Processor Sharing (PS) policy, forwarding its blocks of the requested video to requesting boxes. PS is an adequate model of fair sharing between concurrent TCP connections, when there is no round-trip time bias and the bottleneck is indeed the upstream bandwidth.

We further impose a limit on the number of requests that a box can serve simultaneously. Specifically, to be able to retrieve the video at a rate of R_{enc} (see Table 1), one should receive blocks from each of the $M-1$ target boxes at rate at least R_{enc}/M. Hence we should limit the number of concurrent requests on each box to at most $K_{max} := \lfloor B_{up} * M/R_{enc} \rfloor$, where B_{up} is the upstream bandwidth of each box.

We envision two approaches to handle new video download requests that are blocked because one of the $M-1$ required boxes is already serving K_{max} distinct requests. In the first approach, we simply drop the new request. In the second approach, each of the $M-1$ sub-requests generated by the new request is managed independently at each target box. If the number of concurrent jobs at the target box is below K_{max}, then the sub-request enters service directly. Otherwise, it is put in a FIFO queue that is local to the serving box, and waits there till it can start service.

We refer to the first approach as the blocking model, and to the second as the waiting model.

3.2 Optimality of full striping

We now establish optimality of full striping under waiting

[2]Note that $(M-1)$ boxes are used rather than M because a viewer itself is one of the M boxes.

Table 1: Definition of key parameters

Parameter	Definition
L_i	size of movie i in bytes
M	Total number of boxes in a Push-to-Peer system
R_{enc}	Video encoding rate/playout rate
B_{down}	Downstream bandwidth of each box. We assume B_{down} is infinite, unless otherwise stated.
B_{up}	Upstream bandwidth of each box. We assume $B_{up} < R_{enc}$.
y	maximum number of simultaneous incoming (or outgoing) connections a box can support
W	Size of window in bytes

model. Let us introduce the following stochastic model for demand. Requests for movie j occur at the instants of a Poisson process with rate ν_j. Each such request originates from box m with probability $1/M$, for all $m \in \{1, \ldots, M\}$[3].

Denote by L_j the size of movie j, and by $A_{j,m}$ the amount of memory dedicated to movie j on box m. Then the average size of a download request for movie j is $L_j - (1/M)\sum_{m=1}^{M} A_{j,m}$.

We shall assume that a single copy of each movie is stored in the system, which can be translated into the constraint $\sum_{m=1}^{M} A_{j,m} = L_j$. It is natural to ask whether under such constraints, there exists a placement strategy that is optimal with respect to the demand rates ν_j that it can accommodate. The following shows that full striping is such an optimal placement strategy:

PROPOSITION 1. *Assume that a single copy of each movie is stored in the whole system. Then under full striping data placement, and for the waiting model above described, the system is stable (i.e., download times do not increase unboundedly) whenever the Poisson arrival rates ν_j verify*

$$\sum_{j=1}^{J} \nu_j L_j \,(1 - 1/M) < M * B_{up}. \qquad (1)$$

Moreover, for any other placement strategy specified by the $A_{j,m}$, the set of demand rates ν_j that can be accommodated without rejection is strictly smaller than that under full striping.

PROOF. Note that for any placement policy in which movies are stored only once, the work arrival rate at a given box m is given by

$$\rho(m) := (1 - 1/M) \sum_{j=1}^{J} \nu_j A_{j,m}. \qquad (2)$$

Under full striping, one has $A_{j,m} = L_j/M$. Thus condition (1) is equivalent to the condition that the work arrival rate $\rho(m)$ is less than the service rate B_{up} of box m. This condition does not depend on m, and is thus the necessary and sufficient condition for stability of the whole system.

Consider now a different placement strategy, for which there exists a pair (j^*, m^*) such that $A_{j^*,m^*} > L_{j^*}/M$. For any demand rates ν_j, $j = 1, \ldots, J$, assume that there exists a pull strategy that can stabilise the system under such demand. Then necessarily, for all $m \in \{1, \ldots, M\}$, one has $\rho(m) < B_{up}$. Summing these inequalities one obtains (1), hence such demand can also be handled under full striping.

[3]This assumption of symmetry between boxes can be relaxed. Placement strategy and Proposition 1 below would then need to be suitably modified.

Consider now a particular demand vector where $\nu_j = 0$ for all $j \neq j^*$, and

$$\nu_{j^*}(1 - 1/M)L_{j^*} = MB_{up} - \epsilon,$$

for some small $\epsilon > 0$. Clearly this verifies (1). However, the load placed on box m^* is precisely

$$\rho(m^*) = (1 - 1/M)\nu_{j^*} A_{j^*,m^*}.$$

By our choice of (j^*, m^*), we thus have that

$$\rho(m^*) > (1 - 1/M)\nu_{j^*} L_{j^*}/M.$$

Thus for small enough ϵ, one must have $\rho(m^*) > B_{up}$. Therefore, this box is in overload and the system cannot cope with such demands, while full striping can. □

3.3 Data placement and pull policies under limited number of connections

We describe how the full striping scheme can be extended, given a constraint that the maximum number of simultaneous connections that a box can serve is bounded by y. We propose constrained-striping scheme and coding scheme that limit the maximum number of simultaneous connections to the parameter y.

3.3.1 Constrained-Striping scheme

In constrained-striping scheme, the M boxes are organized in $\lfloor M/(y+1) \rfloor$ disjoint sets. A window of size W is striped into $(y+1)$ blocks of size $W/(y+1)$ and each block is pushed to one single box for each set. A window of a movie can be reconstructed by downloading y distinct blocks for the window in parallel from y boxes in one of the sets.

Consequently, each box in a set keeps a distinct block of a window, and M boxes will have $\lfloor M/(y+1) \rfloor$ copies of movies collectively after the data placement.

3.3.2 Coding scheme

In addition to constrained striping, we also propose a coding scheme that applies rateless coding [16, 15]. It is known that rateless codes such as LT code [15] can generate infinite number of unique so called coded symbols by combining the k source symbols of the original content. For decoding any set of $(1+\epsilon)*k$ distinct coded symbols are needed, and the k source symbols can be reconstructed with high probability. In practice, the overhead parameter ϵ can be in $[0.03, 0.05]$, depending on the specific coding that we use [16, 5].

For the specific case of Push-to-Peer, to ensure that a peer can reconstruct a window of size W by downloading from y different boxes, each box has to serve a fraction of size $W * (1 + \epsilon)/(y + 1)$ of the window. y is the maximum

number of simultaneous connections that a box can support as described previously.

The coding scheme we propose consists in dividing each window into k source symbols[4], and generate $C * k = (M * (1+\epsilon)/(y+1)) * k$ coded symbols. We call C the expansion ratio, where $C > 1$. For each window, the $C * k$ symbols are evenly distributed to all M boxes such that each box keeps $C * k/M = (1+\epsilon) * k/(y+1)$ distinct symbols. A viewer can reconstruct a window of a movie by downloading any $C * k * y/M$ distinct symbols *in parallel* from arbitrary a set of y boxes out of $(M-1)$ boxes.

The coding scheme is similar to full-striping scheme in the sense that distinct (coded) symbols are striped to all M boxes. However, unlike full-striping scheme, we only need to download from any y boxes.

3.3.3 Pull policy for coding scheme

We now define the pull strategy used for the coding scheme. We assume a maximum number of requests, K, can be processed concurrently on each box. Each box has a queue from which it selects requests that will be treated by the box.

Movie download requests are broken into y sub-requests, that occupy consecutive slots in the queue of each box, except for the box that issued the request. When a box becomes available to serve a new (sub-)request, it selects the one closest to the head of its queue, and for which it has not started serving another sub-request that is part of the same global request. Once a sub-request has been selected by a box, the sub-request is removed from the queue of all other boxes. Each box then uses Processor Sharing among currently handled sub-requests.

3.4 Optimality of coding scheme

We assume additional storage is used per movie as described before. Specifically, we assume that a total storage capacity of $C*L_j$ is devoted to movie j, where C corresponds to expansion ratio introduced in the previous section. The solution based on encoding assumes that for movie j, a total quantity of $A_{j,m} \equiv C * L_j/M$ data is stored on each individual box m. This data consists of symbols, such that for any collection of $y+1 = M/C$ boxes, each movie can be reconstructed from the joint collections of symbols from all these $y+1$ boxes. We then have the following proposition:

PROPOSITION 2. *By using the pull strategy described in section 3.3.3, the system is stable whenever the Poisson arrival rates ν_j verify*

$$\sum_{j=1}^{J} \nu_j L_j \, (1 - C/M) < M * B_{up}, \tag{3}$$

and this is the best possible for the amount of memory used by the system.

In the interest of space, we omit the proof, which will appear in a companion technical report. We only note that the average amount of data that needs to be downloaded for a request for movie j is $L_j(1-C/M)$ when the overall storage devoted to movie j is CL_j, and hence the left-hand side of (3) is indeed the rate at which work enters the system, while the right-hand side is an upper bound on the service capacity of the system. Thus with the assumed total storage per movie, Condition (3) is indeed necessary to ensure the existence of a pull strategy for which the system is stable.

[4]With rateless codes the greater k, the greater is the probability to reconstruct content with small overhead [5]. Consequently, the symbol size should be as small as possible, and therefore in our case symbol size should be equal to packet size (i.e. MTU).

4. PERFORMANCE ANALYSIS

4.1 Blocking model

We now propose simple models to predict the blocking probability of the system in the blocking model for the distinct placement strategies we introduced.

We first consider full striping. In the actual system, the number of requests in progress can vary from box to box, essentially because a requesting box does not place a request on itself. Also, the overall service speed varies between $(M-1)B_{up}$ and $M * B_{up}$ depending on the system state: when a single video download is taking place in the whole system, it proceeds at speed $(M-1)B_{up}$, while an overall service rate of $M * B_{up}$ is achieved when sub-requests are served on all boxes.

However we consider simplified dynamics, where the number of sub-requests is the same on each box, and where the total service speed is also constant. Specifically, we consider a total service capacity of $B_{total} = M * B_{up}$ and assume this is shared evenly among active downloads. The total amount of data that needs to be downloaded for movie j playback is then taken to be $L_j(1-1/M)$, as in the actual system. We assume Poisson arrival rate ν_j of download requests for movie j, and a maximum number of concurrent downloads of $K_{max} = \lfloor B_{total}/[R_{enc}(1-1/M)] \rfloor$. These simplified dynamics correspond to the classical M/G/1/K/PS model, the blocking probability of which is given by (see e.g. [13])

$$P_b^I := \frac{(1-\rho)\rho^{K_{max}}}{[(1-\rho^{K_{max}+1})]} \tag{4}$$

where

$$\rho = \frac{\sum_{j=1}^{J} \nu_j L_j (1-1/M)}{B_{total}} \, . \tag{5}$$

To model the performance under either constrained striping or coding schemes we make similar simplifying assumptions. We again assume that each box handles the same number of sub-requests, so that the system state is captured by the total number of movie download requests. However we take into account the fact that each movie request is served by a maximum of y boxes, by taking the total service rate, when there are n movie requests, as the minimum of B_{total} and $n * B_{down}$ where $B_{down} := y * B_{up}$.

Under such simplifying assumptions, the system state evolves as a birth and death process on $\{0, \ldots, K_{max}\}$, where now $K_{max} = \lfloor B_{total}/[R_{enc}y/(y+1)] \rfloor$. The birth rate equals $\nu = \sum_j \nu_j$ in all states except K_{max}, and the death rate in state n is given by[5]

$$\frac{\min(n * B_{down}, B_{total})}{\overline{\sigma}}$$

[5]We would indeed have a Markovian birth and death process if job sizes were exponentially distributed, and with mean $\overline{\sigma}$. Insensitivity results on Processor Sharing systems, see e.g. [12] guarantee that the rejection probability is insensitive to the actual service time distribution and justify formula (6) for the case of mixtures of deterministic service time distributions.

Table 2: Parameters for analysis

Parameter	Value
Total number of boxes (M)	512
minimum block size	1024 bytes
Video encoding rate (R_{enc})	2Mbps
Upstream bandwidth (B_{up})	1Mbps
Downstream bandwidth (B_{down})	infinite
Size of Video (L)	2Gbytes
maximum number of simultaneous incoming connection (y)	31

where $\overline{\sigma}$ is the average job size. This reads

$$\overline{\sigma} = \sum_j (\nu_j/\nu) L_j(y/(y+1))$$

in the case of constrained striping; we inflate this number by $(1+\epsilon)$, with $\epsilon = 5\%$, to reflect coding overhead when considering coding.

For this system the blocking probability, which coincides with the probability to be in state K_{max} in steady state, reads

$$P_b^{II} := \frac{\rho_1^k/k!\ \rho_0^{K_{max}-k}}{\sum_{i=0}^{k} \rho_1^i/i! + (\rho_1^k/k!)(1-\rho_0^{K_{max}-k+1})/(1-\rho_0)} \tag{6}$$

where we have introduced the notations

$$\begin{aligned} \rho_0 &:= \tfrac{\nu\overline{\sigma}}{B_{total}},\ \rho_1 := \rho_0 \tfrac{B_{total}}{B_{down}}, \\ k &:= \lfloor \tfrac{B_{total}}{B_{down}} \rfloor. \end{aligned}$$

The derivation is a simple exercise, and is omitted for brevity.

Table 2 describes the parameters we use for numerical evaluation of the above formulas. We plot the rejection rate for each data placement scheme in Figure 3 using the parameters. Because of upstream bandwidth limitation (i.e., $R_{enc}/B_{up} = 2$), the maximum number of viewers that can be admitted into the system is at most 256. The x-axis indicates the normalized arrival rate of user requests and the y-axis indicates the rejection probability of user requests.

Over a wide range of arrival rates, the rejection rates do not differ much between all three schemes. Perhaps surprisingly, the full striping scheme consistently outperforms constrained striping and coding schemes, even though the last two schemes benefit from larger amounts of data stored on each box. Compared to constrained striping and coding schemes, full striping allows viewers to take advantage of the bandwidth from all M boxes regardless of the number of served viewers. Rather, in constrained striping and coding schemes, the maximum bandwidth share that a viewer would get is at most $y * B_{up}$ regardless of the number of served viewers. Because of ϵ overhead, the coding scheme performs slightly worse than constrained striping.

4.2 Waiting model

In this section we consider performance of the system under the waiting model. As for blocking, we make simplifying assumptions to define a tractable performance model. Specifically, we again assume that all boxes handle the same numbers of sub-requests. Thus, an incoming movie request is either accepted on all boxes, in which case it gets a fair

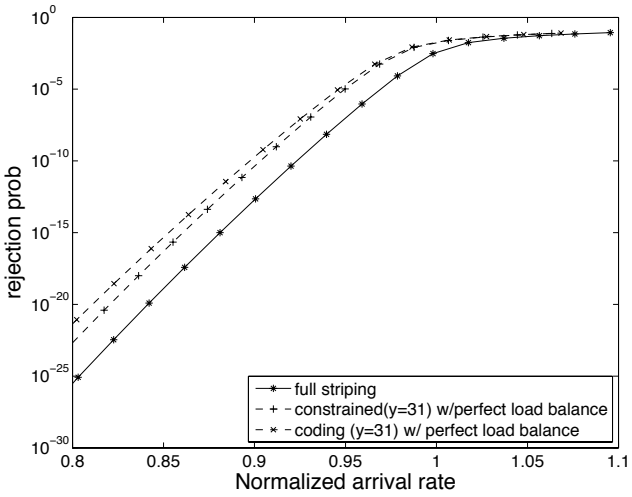

Figure 3: rejection Probability

share of the overall system upstream bandwidth, if there are less than K_{max} jobs in the system. Otherwise, the job is put in a single FIFO queue. Again K_{max} is determined to ensure that effective download rate is at least playback rate R_{enc}.

We call this system the FIFO+PS service system. While its performance is well understood under the assumptions of Poisson job arrivals and exponential service times, to our knowledge its performance has not been analysed previously when the assumption of exponential service times is relaxed. One of our contributions is to provide such an analysis, in a heavy traffic regime.

Notations are as follows. Service capacity is normalised to 1. Jobs arrive at instants of a Poisson process with intensity ν_ℓ. Jobs are i.i.d. with some fixed distribution; we denote by σ a typical job service time. The number of jobs that can be served concurrently is denoted K_ℓ. The index ℓ is introduced to set the stage for the heavy traffic analysis. Indeed, denoting by $\rho_\ell := \nu_\ell \mathbf{E}(\sigma)$ the traffic intensity, we shall assume that, as ℓ tends to infinity, the load approaches 1 from below:

$$\rho_\ell < 1,\ \ell \geq 1; \quad \lim_{\ell\to\infty} \rho_\ell = 1.$$

We shall further assume the existence of a positive number m such that:

$$\lim_{\ell\to\infty} (1-\rho_\ell) K_\ell = m.$$

We then have the following result, the proof of which is deferred to the appendix:

THEOREM 1. *Assume that the service time distribution is a finite mixture of Exponential distributions. Denote by Z^ℓ the number of jobs in steady state in the ℓ-th system. One then has the following convergence, for all $t > 0$:*

$$\lim_{\ell\to\infty} \mathbf{P}\left((1-\rho_\ell) Z^\ell > t\right) = \begin{cases} e^{-m-2(t-m)\overline{\sigma}^2/\overline{\sigma^2}} & \text{if } t > m, \\ e^{-t} & \text{if } t \leq m. \end{cases} \tag{7}$$

Furthermore, denoting by W^ℓ the waiting time of a job in steady state in the ℓ-th system, one has the following con-

vergence, for all $t \geq 0$:

$$\lim_{\ell \to \infty} \mathbf{P}\left((1-\rho_\ell)W^\ell > t\right) = e^{-m-2t\overline{\sigma}/\overline{\sigma^2}}. \quad (8)$$

In particular the probability of not waiting satisfies

$$\lim_{\ell \to \infty} \mathbf{P}(W^\ell = 0) = 1 - e^{-m}. \quad (9)$$

REMARK 1. *Although we have established the theorem only for the case of service times that are mixtures of exponential distributions, we expect it to hold more generally.*

We now indicate how to use this result. For given system parameters, we approximate the distribution of the waiting time of an arbitrary job as follows:

$$\mathbf{P}(W^\ell > t) \approx e^{-(1-\rho_\ell)[K_\ell + 2t\overline{\sigma}/\overline{\sigma^2}]}. \quad (10)$$

4.3 Application: sizing prefixes

We now show how to use the previous results to further optimize content placement assuming extra storage is available. We again assume there are J movies, all encoded at a constant bit rate R_{enc}, and denote by L_j the size of movie j.

For movie j, we assume that a prefix of size P_j is stored locally on each box. This ensures that each user can play back movie the first $t_j := P_j/R_{enc}$ seconds of movie j without downloading extra content. We further assume that encoded symbols are created and placed on each box so that for each movie j, its remainder can be reconstructed from the symbols present at any $y+1$ boxes.

Let D denote the memory space available on each box. The above described placement strategy will be feasible provided the constraint below is satisfied:

$$\sum_{j=1}^{J} P_j + (L_j - P_j)/(y+1) \leq D. \quad (11)$$

Denote by ν_j the rate of requests for movie j. The amount that needs to be downloaded for playback of movie j is then

$$\sigma_j = \frac{y}{y+1}(L_j - P_j). \quad (12)$$

Indeed, the prefix of size P_j is stored locally, as well as a fraction $1/(y+1)$ of the remainder of the movie. The normalised load on the system is thus:

$$\rho = \frac{\sum_{j=1}^{J} \nu_j \sigma_j}{B_{total}}. \quad (13)$$

4.3.1 Blocking probabilities

We first consider performance under blocking. The maximum number of concurrent jobs is $K_{max} = \lfloor B_{total}/[R_{enc}y/(y+1)] \rfloor$. Blocking probability is given by Formula (4) in the particular case where $y+1 = M$, that is to say under full striping. This probability is then minimised by making the load as small as possible.

As is easily seen, to minimise the load ρ as given by (13) and (12) under memory constraints (11) one should aim to cache locally the most popular movies in full.

4.3.2 Waiting times

We now assume FIFO queueing rather than rejection of requests initiated at times where number of concurrent requests equals N.

The evaluations (10) give us an approximation of the distribution of the delay W between request initiation and download beginning. The actual delay can be reduced because playback can start t_j seconds before download starts.

This yields the following expression for the average actual delay $\bar{D}_j$ experienced by requests for movie j:

$$\begin{aligned} \bar{D}_j &= E\left[\max(0, W - t_j)\right] \\ &= \int_{t_j}^{\infty} (x - t_j) \frac{2\overline{\sigma}(1-\rho)}{\overline{\sigma^2}} e^{-(1-\rho)[K_{max} + 2x\overline{\sigma}/\overline{\sigma^2}]}. \end{aligned}$$

We thus obtain the formula

$$\bar{D}_j = \frac{\overline{\sigma^2}}{2\overline{\sigma}(1-\rho)} e^{-(1-\rho)[K_{max} + 2t_j\overline{\sigma}/\overline{\sigma^2}]}. \quad (14)$$

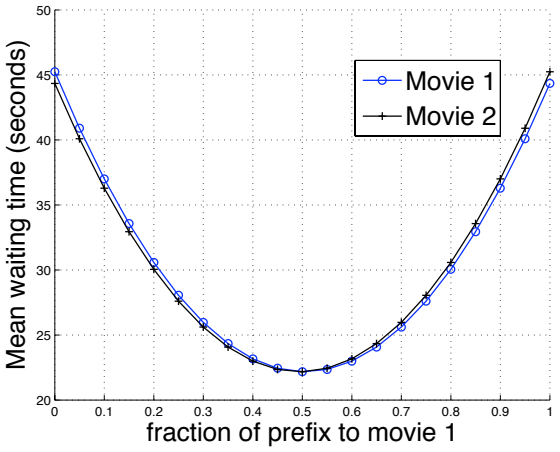

Figure 4: waiting time against prefixes balanced popularity ($\nu_1 = \nu_2 = 0.99$, $R_{enc} = B_{total} = 1$, $L_1 = L_2 = 1$, $P_1 + P_2 = 1$, $y \gg 1$)

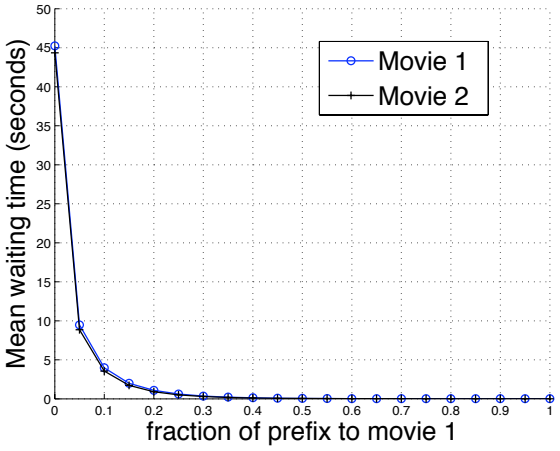

Figure 5: waiting time against prefixes distinct popularity ($\nu_1 = 0.99$, $\nu_2 = 0.5$, $R_{enc} = B_{total} = 1$, $L_1 = L_2 = 1$, $P_1 + P_2 = 1$, $y \gg 1$)

We use a simple example to illustrate how a fixed amount of memory in a box can be optimally allocated to movies with same or distinct popularities to preload prefixes of

movies. Figures 4 and 5 show plots of the mean waiting times $\overline{D}_j$ obtained from Formula (14). In each case, there are two movies, and there is a fixed amount of memory that can be used for prefixes of one movie or the other. In Figure 4, the popularities of both movies are same. The figure indicates that in case of balanced popularity for two movies, the best for both movies is to get equal prefixes. Note that varying prefixes does not change load here. Also, it does not change the average service time $\bar{\sigma}$. So it would seem that one movie would benefit from having a larger prefix. It is however not the case, because unbalanced prefixes lead to large variance of service times and thus large second moment $\overline{\sigma^2}$.

In Figure 5, we have distinct popularities, with movie 1 roughly twice as popular as movie 2. The figure indicates that it is beneficial to both movies to have the prefix memory devoted to movie 1. Here, by storing large prefixes for movie 1, we reduce the system load ρ, and this is the leading effect.

5. THE CASE OF UNCONTROLLED ENVIRONMENTS

In this section, we discuss adaptations of our techniques to make them applicable to uncontrolled environments. These concern mostly the pull phase, and we assume throughout this section that original content placement has been done according to the coding strategy previously defined in Section 3.

We first propose a distributed randomized policy for initial job placement. This does not rely on a central control server, which was necessary to implement the job placement strategies we proposed for controlled environments in Section 3.

We also discuss how to cope with changes in uplink bandwidths of boxes. In an environment where boxes are controlled by an ISP, the ISP can guarantee that a fixed amount of bandwidth is devoted to the Push-to-Peer service. In uncontrolled environments such as the public Internet, the Push-to-Peer service may compete with other application traffic on the Internet, and available bandwidth may thus change. We show how one can hedge against such changes by downloading from more boxes than strictly necessary – thereby exploiting the flexibility enabled by content coding– and also propose a job migration strategy to reconfigure job placement after a change in bandwidths.

5.1 Randomized job placement

The strategy we consider for initial job placement is as follows. When a download request is generated, d distinct boxes are chosen at random from the overall collection of M boxes. The load, measured in terms of fair bandwidth share that a new job would get, is measured on all probed boxes. Finally, sub-requests are placed on the y least loaded boxes among the d probed boxes, provided that each of the y sub-requests gets a sufficiently large fair bandwidth share, i.e. larger than or equal to $(y/(y+1))R_{enc}$ with our previous notation. If any of the least loaded boxes cannot guarantee such a fair share, then the whole request is dropped.

We assume in the present sub-section that each box has a fixed overall upstream bandwidth of B_{up} (this is relaxed in the second half of the section). Thus the maximum number of sub-requests on each box is $K_{max} = \lfloor B_{up}/[y/(y+1)R_{enc}]\rfloor$.

Table 3: Definition of key parameters

Parameter	Definition
M	Total number of jobs
K_{max}	Maximum queue size for each box
y	Number of parallel sub-jobs
d	Number of boxes probed
μ	service rate of a processor
$M\lambda/y$	Arrival rate of jobs
p_i	fraction of boxes with i sub-jobs

Many results are available on the performance of related randomized load balancing schemes. If we assume Poisson arrivals of requests at rate $\lambda * M/y$, no rejection ($K_{max} = \infty$), $y = 1$ (requests generate a single sub-job) and exponential job size distribution, we have exactly the model analyzed by Vvedenskaya et al. [24] (see also Eager et al. [8] and Mitzenmacher et al. [18]). For this system they show that, in the large M limit, in steady state the fraction ϕ_i of all M boxes that contain at least i jobs is given by

$$\phi_i = \rho^{\frac{d^i-1}{d-1}},$$

where ρ is the normalized load on each box.

The system we consider differs by the fact that there are several sub-jobs, and by the possibility of job rejection. It is however amenable to a similar analysis. We now determine fixed point equations that characterize the fraction of boxes holding a given number of sub-jobs in equilibrium. We do not claim the derivation is rigorous, but instead validate it by simulations.

Notations used in the analysis are summarized in Table 3. The heuristic derivation proceeds as follows. Fix $i \in \{0, \dots, K_{max}\}$. For a new request, denote by $X_{<i}$ (respectively X_i, $X_{>i\&<K_{max}}$ and $X_{K_{max}}$) the number of sampled boxes with less than i jobs (respectively i, more than i and less than K_{max} and K_{max}). The vector of these four quantities follows a multinomial distribution with parameters d and $(p_{<i}, p_i, p_{>i\&<K_{max}}, p_{K_{max}})$, where

$$p_{<i} := \sum_{j<i} p_j,\ p_{>i\&<K_{max}} := \sum_{i<j<K_{max}} p_j.$$

Denote by $F_i(u, v, w, z)$ the probability that this multinomial distribution puts on the vector (u, v, w, z). Its dependence on the parameters p_j is not made explicit to simplify notation. Denote by G_i the expected number of boxes which previously had i jobs and receive a new one from such a new request. Then this can be written as

$$G_i = \sum_{u,v,w,z} F_i(u, v, w, z) \min(v, \max(0, y - u))\ \mathbf{1}_{z \le d-y}.$$

Indeed the factor $\mathbf{1}_{z<d-y}$ retains only terms in the summation where all sub-jobs can get enough bandwidth, and the term $\min(v, \max(0, y - u))$ counts the number of least loaded boxes that currently have i jobs. With such notation at hand write the heuristic differential equation:

$$\frac{d}{dt} M p_i = M(\lambda/y)(G_{i-1} - G_i) - \mu M(p_i - p_{i+1}).$$

The rationale is that new boxes with i jobs appear at rate $M\lambda/yG_{i-1}$ because of extra jobs being placed on boxes previously holding $i-1$ jobs, and also at rate $\mu M p_{i+1}$ because

of departures from boxes previously holding $i+1$ jobs. The rationale for departure rates is similar.

The fixed point equation for p_i is then obtained by setting the right-hand side of the previous equation to 0. Introduce now the notation

$$\lambda_i := \frac{\lambda}{y}\frac{G_i}{p_i}.$$

The fixed point equations may then be written as

$$p_{i+1}\mu = \lambda_i p_i,\ i = 0, \ldots, K_{max} - 1.$$

Since $\sum_{i=0}^{K_{max}} p_i = 1$, we obtain in turn

$$p_0 = \frac{1}{1+\sum_{i=1}^{K_{max}}[(\prod_{j=0}^{i-1}\lambda_j)/\mu^i]} \quad (15)$$

$$p_n = \frac{\prod_{j=0}^{n-1}\lambda_j}{\mu^{n-1}} p_0,\ n = 0, \ldots, K_{max}. \quad (16)$$

Note that the parameters λ_i in the right-hand sides of these expressions depend on the p_i's themselves. The fixed point equations (15), and (16) cannot be solved explicitly. However we obtain a numerical approximation by applying iteratively the function specified by (15), and (16) on an initial guess for the p_i's. We observed fast numerical convergence of the iterations in our experiments. Once the parameters p_j are determined, the rejection probability is determined according to the formula

$$p_{reject} = \sum_{i=d-y+1}^{d} \binom{d}{i} p_{K_{max}}^i (1 - p_{K_{max}})^{d-i}.$$

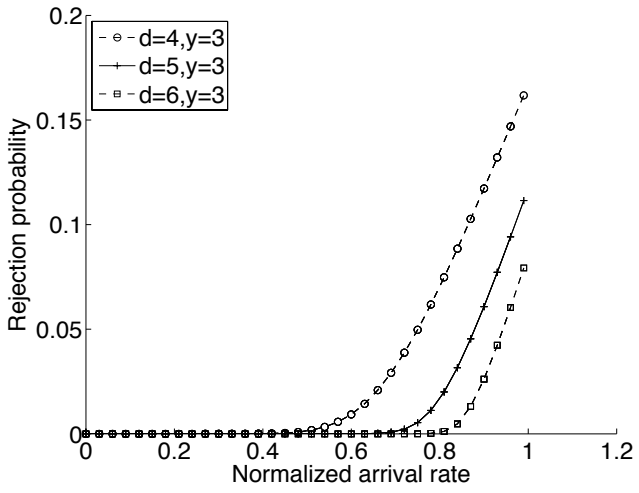

Figure 6: Numerical solutions for rejection probability using the proposed load balancing scheme

Figure 6 shows the results we obtain for distinct choices of parameters (y, d) for varying normalized load $\rho = \lambda/\mu$, and setting K_{max} to 3.

Figure 7 is obtained by simulation, using $M = 50$ boxes. The results match reasonably well those in Figure 6. We believe that the fixed point equations we just described are accurate in the large M limit.

More importantly, we observe that even at normalised loads as high as 100%, the rejection probabilities remain small: below 15% when only one additional box is probed and down to 5% when three additional boxes are probed.

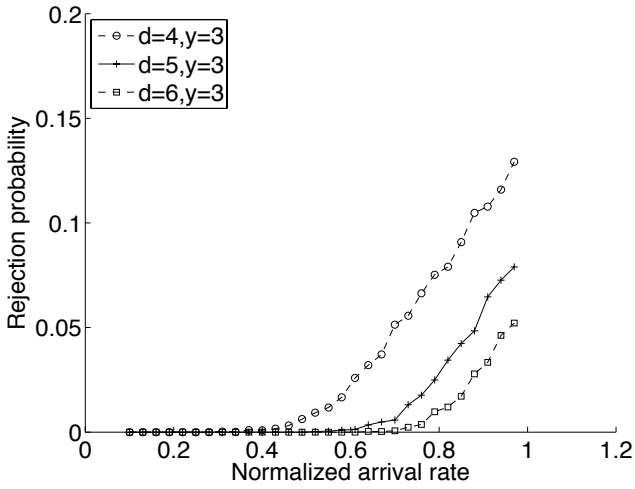

Figure 7: Simulation result for rejection probability using the proposed load balancing scheme

5.2 Handling upstream bandwidth variability

This section discusses ways to cope with changes in available upstream bandwidth. More precisely, we address the following two possible scenarios: (a) *Upstream bandwidth diversity*: Upstream bandwidth of a box is consistently smaller than upstream bandwidth of other boxes during a sufficiently large time period. (b) *Upstream bandwidth fluctuations*: The average bandwidth of a box does not change, however, the fluctuation increases the variability of upstream bandwidth.

We propose two complementary strategies to cope with the above two problems.

The first strategy called *overbooking* strategy, addresses bandwidth diversity and allows a viewer to download more symbols from boxes having large bandwidth, by placing extra symbols in each box in push phase. So far we have assumed that we download content from the smallest possible number of boxes, from which content can be decoded. In that case, effective download rate is constrained by the slowest box. Overbooking basically amounts to download from more boxes than strictly needed, either by increasing the limit on the maximum number of connection, or by increasing the amount of coded data pushed on the boxes.

Overbooking may also be used to overcome bandwidth fluctuations: by staying long enough with a set of boxes, we absorb the fluctuations over time. However, to apply such policy a precise characterization of bandwidth fluctuations is needed. Having a precise bandwidth fluctuation model is difficult and we therefore rather propose the following strategy that is complementary to the overbooking strategy.

The second strategy called *job-migration* strategy, addresses bandwidth fluctuations and consists in migrating a viewer's request from the box serving the request to another box having more available uplink bandwidth, when the downloading rate goes below some threshold.

We calculate the time it takes to reconfigure the system under bandwidth fluctuation. The timescale of bandwidth fluctuation should be larger than the time to reconfigure the system, that is in fact a function of probing frequency. Note also that our analysis also allows to dimension the playout

buffer, since this one should be larger than the time it takes to reconfigure the system.

5.2.1 Overbooking strategy

We now consider the issue of overbooking resources in order to cope with fluctuating bandwidth. Rather than downloading exactly equal number of symbols in parallel from y boxes, a viewer can be allowed to download more symbols from the boxes with larger upstream bandwidth than from the boxes with smaller upstream bandwidth. It can be made possible by placing more symbols (preferably in as small size as possible) to each box in push phase.

More specifically, assume each box contains a quantity $(C/M)W$ of symbols of data for each data window of size W. Assume symbols are small enough so that their granularity can be ignored. Let a peer download data from y other peers, with $y \geq z = M/C-1$, and let the resulting download speeds be $v_1 \geq v_2 \geq \ldots \geq v_y$. z is the minimum number of boxes that a viewer needs to contact to reconstruct the window. We now determine the overall speed at which data can be recovered.

The time T needed to reconstruct the original window of size W must be such that

$$\sum_{i=1}^{y} \min((1/(z+1))W, v_i T) = (1-1/(z+1))W.$$

Indeed, the minimum in the summation term represents the amount of data from the window of interest that can be downloaded from the i-th peer in time T. Given a bandwidth diversity model, the above characterization should allow to determine the probability that window download is to slow. In turn it can inform selection of parameters z and C.

5.2.2 Job migration strategy

We now consider the issue of rebalancing load after a change of available bandwidth resources at the boxes.

The precise scenario is as follows. The M boxes have available uplink capacities C_i, $i = 1, \ldots, M$. Initially there are N_i jobs on box i for all $i = 1, \ldots, M$. Load balancing operates as follows. Each job, at the instants of a Poisson process of rate β, selects a box uniformly at random from all available boxes. Assume the job was initially placed on box i, and samples box j. Then it migrates to box j if and only if the following two conditions are met. First, it will get a larger bandwidth share by doing so, that if $C_i/N_i < C_j(N_j+1)$. Second, by migrating, it won't let the fair shares obtained at the target box j go from above the playback rate R_{enc} to below R_{enc}, i.e. migration is prevented if $C_j/N_j \geq R_{enc} > C_j/(N_j+1)$.

It should be clear that the capacity profile affects the speed at which this scheme will manage to balance the allocations obtained by each job. Indeed, if all capacities C_i equal zero except for the capacity C_1 of the first box, perfect load balancing is achieved by migrating all jobs to box 1. However each job has a probability of only $1/M$ of discovering the correct box, hence a very long time to even approximate load balancing when the number of boxes, M is large.

With some control on the imbalance of the capacity profile C_i, it is nevertheless possible to derive some results on the speed at which such load balancing performs. We will more specifically be concerned with the time till all jobs get a target bandwidth share of R_{enc}[6]. To this end introduce the following notations:

$$\begin{cases} K_i & := \lfloor C_i/R_{enc} \rfloor,\ i = 1, \ldots, M, \\ \delta & := 1 - \frac{\sum_{i=1}^{M} N_i}{\sum_{i=1}^{M} K_i}. \end{cases}$$

In words, K_i is the number of jobs that can be offered a bandwidth of R_{enc} at box i, and δ characterises the fraction of the number of extra jobs that could be supported by the system while still getting a target capacity of R_{enc} to the total number of jobs that can be provided such target capacity.

We then have the following simple result:

LEMMA 1. *Assume that δ is strictly positive. Then the fraction of boxes which could host an additional job and still provide a bandwidth share of R_{enc} to all supported jobs is at least*

$$f := \delta \frac{\overline{K}}{\max_{i=1}^{M} K_i}, \tag{17}$$

where $\overline{K}$ is the average of the quantities K_i over all boxes i.

PROOF. By definition of δ, the total number of jobs that could be offered a bandwidth of R_{enc} given the current capacity profile $\{C_i\}_{i=1}^{M}$ reads

$$\sum_{i=1}^{M} K_i = \sum_{i=1}^{M} N_i + \delta \sum_{i} K_i.$$

Thus, for any specific placement of the jobs, there are at least $\delta \sum_i K_i$ extra jobs that could be accepted in the system and still get a bandwidth share R_{enc}. However, no more than $max_i K_i$ such jobs could be placed on any box i, by definition of K_i. Hence it follows that the number M^* of boxes that can accept an additional job and still provide a fair share of at least R_{enc} to all its jobs verifies

$$M^* \max_{i=1}^{M} K_i \geq \delta \sum_{i=1}^{M} K_i,$$

from which Equation (17) follows. □

It is interesting to rewrite expression f as follows:

$$f = \frac{\overline{K} - \overline{N}}{\max_{i=1}^{M} K_i}.$$

It then becomes apparent that the probability that a randomly selected box can host an additional job while still providing a bandwidth of R_{enc} is at least equal to the ratio of the average number of extra jobs that could be added per box, to the maximal number of jobs that can be supported by any box.

This result allows us to obtain upper bounds on the time for the system to reconfigure after a change in available bandwidth capacities:

PROPOSITION 3. *Under Lemma 1's assumptions, the time for any job to successfully migrate to a box where it receives a bandwidth share of R_{enc} is not larger than an Exponential*

[6]Note that this analysis assumes a job-size of R_{enc}, i.e. we do not consider a job is broken down into y sub-request. Since this does not really affect the results of this analysis, we keep it to the reader as a simple exercise to do the analysis in the case with sub-request.

random variable with parameter βf, where β is the probing rate, and f is defined in (17).

Let N_{bad} denote the number of jobs that initially receive less than R_{enc}. Then assuming fixed β and f, in the limit when N_{bad} becomes large, the time T_{reconf} to successful replacement of all jobs satisfies

$$T_{reconf} \leq \frac{1}{\beta f} \log(N_{bad}) + O(1). \tag{18}$$

The time $T_{reconf}(x)$ to successful replacement of a fraction x of jobs reads

$$T_{reconf}(x) \leq \frac{1}{\beta f} \log\left(\frac{1}{1-x}\right) + o(1). \tag{19}$$

PROOF. The first statement of the proposition is a direct consequence of Lemma 1: each job probes at rate β, and each such probe is to a box that can host it and provide a bandwidth R_{enc} with probability at least f. By erasing with probability $(1-f)$ the time points of a Poisson process of rate β, one obtains a Poisson process with rate βf. Thus the time till a "good" box is probed is at most the first instant of a Poisson process with rate βf, hence at most an Exponentially distributed random variable with parameter βf.

To establish (18), let T_n denote the time to successful reconfiguration of the n^{th} initially badly placed job, for $n = 1, \ldots, N_{bad}$. By the first part of the proposition, one has for all $x \in R$:

$$\begin{aligned} P\left(\sup_{n=1}^{N_{bad}} T_n > \tfrac{\log(N_{bad})+x}{\beta f}\right) &\leq 1 - \left[1 - e^{-\log(N_{bad})-x}\right]^{N_{bad}} \\ &= 1 - \left[1 - \tfrac{1}{N_{bad}} e^{-x}\right]^{N_{bad}} \\ &\to 1 - \exp(-e^{-x}) \end{aligned}$$

as $N_{bad} \to \infty$. This establishes (18) (it should be understood in this expression that the O(1) term is in fact a random variable; this is sometimes referred to as "O(1) in probability").

To establish (19), note that for any $\epsilon > 0$, the probability that $T_{reconf}(x)$ is less than $1/(\beta f)[\log(1/(1-x)) + \epsilon$ is larger than the probability that a binomial random variable with parameters N_{bad} and

$$1 - \exp(-\epsilon - \log(1/(1-x))) = 1 - (1-x)e^{-\epsilon}$$

is larger than xN_{bad}. Standard properties of Binomial distribution entail that this holds with probability close to 1 as $N_{bad} \to \infty$, and the result (19) follows. □

6. RELATED WORK

The use of peer-to-peer networks for streaming video on the Internet is a topic that has received some recent attention [6, 26, 19, 23]. However, most of the efforts have focused on efficient tree and mesh construction assuming the upstream bandwidths of many or all of peers are strictly larger than video playback rate. This naive assumption enables those p2p systems to scale to support arbitrary large number of clients. On the contrary, we assume that upstream bandwidths of all peers are strictly smaller than video playback rate, that is in fact true in most of access networks, particularly in DSL networks that we are focusing on. More recently, Dana et al. [7] and Tewari et. al [22] proposed BitTorrent-based live streaming service under the same assumption of limited upstream bandwidth. In both proposals, the upstream bandwidth limitation is overcome by the assistance of server-based stream delivery in their proposed systems. However, the Push-to-Peer system does not assume the support from the content server after content is proactively pushed to peers and persistently stored before the actual peer-to-peer transfers.

Load balancing strategies have also been investigated in the context of job scheduling in distributed systems and more general bins and balls problems [8, 17, 18]. To the best of our knowledge, all of the proposed load balancing schemes are targeted to balance loads of independent jobs. On the contrary, we address the problem of balancing load of sub-requests from a job, that should be co-scheduled ideally. More recently, the load balancing in case of bulk arrivals of jobs has been investigated by Adler et al. [1], however, the balancing decision is made per job rather than per the set of jobs arriving together. Our proposed scheme collectively balance all sub-requests for a job.

Another related area of work is the data placement and pull scheme for video streaming services. Several methods have been proposed in the literature [14, 20, 21]. Particularly, random duplicated assignment strategy of data blocks is proposed for VoD servers by Korst [14] to address the problem of disk failure. However, we use a coding scheme that addresses the problem of box failures. The prefix prefetching scheme for p2p video streaming [20, 21] cannot be directly applicable to our case because it requires that upstream bandwidth of a peer should be strictly larger than video playback rate.

Rateless coding schemes have been proposed by [5, 16, 15]. While these works discuss how to use the codes to download files using multicast/broadcast transmissions [5] or using peer-to-peer networks [16], none of these works address the usage of coding for video streaming or video-on-demand. Other work proposed the usage of network coding to accelerate file download in peer-to-peer networks [9] or to ameliorate VoD for p2p [2]. Because of the push-phase, our approach does not require that peers serve content that they downloaded previously from other peers. Therefore the usage of network coding is not appropriate to our scenario.

7. CONCLUSION AND FUTURE WORK

We proposed Push-to-Peer, a novel peer-to-peer approach to cooperatively stream video using push and on-demand pull of video contents. We showed the theoretical upper performance bounds that are achieved if all resources of all peers are perfectly pooled, and present the placement (namely full-striping and coding scheme) and pull policies that achieve those bounds. However, perfect pooling is only possible with global knowledge of system state, which in practice is not feasible. Therefore, we discussed several distributed load balancing algorithms adapted to either fixed or fluctuating bandwidth. Rather than giving precise performance results, we propose models and analyses that allow to have insights on the behavior of each of those algorithms.

Future work will address more precise bandwidth fluctuation models, that take into account user behavior and the scale of the bandwidth fluctuation. These models may lead us to refine our present analysis and strategies, as well as identify new strategies that can cope with such fluctuations.

We would like to extend our analysis to include the case where multiple movies of different popularity are served by the system. Even if our first analysis discusses the impact of prefix attribution to different movies regarding blocking

probability and waiting time, we think that there is space for improvement. The non-uniform size of prefixes preloaded for different movies makes the use of processor sharing scheduling less effective, because the deadline for downloading a window is determined by the size of preloaded prefix. To address this issue, we plan to adopt Earliest Deadline First (EDF) scheduling policies developed for multiprocessors.

Finally other features like VCR functionality support or multicast assistance for downloads needs to be investigated.

8. REFERENCES

[1] M. Adler, S. Chakrabarti, M. Mitzenmacher, and L. Rasmussen. Parallel randomized load balancing. In *Proc. ACM Symposium on Theory of Computing (STOC)*, pages 238–247, 1995.

[2] S. Annapureddy, C. Gkantsidis, and P. Rodriguez. Providing Video-on-Demand using Peer-to-Peer networks. In *Proc. Internet Protocol TeleVision (IPTV) workshop*, 2006.

[3] F. Baccelli and P. Brémaud. *Elements of Queueing Theory*. springer, 1994.

[4] P. Bratley, B. Fox, and L. Schrage. *A guide to simulation*. Springer-Verlag, 1986.

[5] J. W. Byers, M. Luby, and M. Mitzenmacher. A digital fountain approach to asynchronous reliable multicast. *IEEE J-SAC, Special Issue on Network Support for Multicast Communication*, 20(8), 2002.

[6] Y. Chu, S. Rao, S. Seshan, and H. Zhang. Enabling conferencing applications on the internet using an overlay multicast architecture. In *Proc. ACM SIGCOMM*, 2001.

[7] C. Dana, D. Li, D. Harrison, and C. Chuah. BASS: BitTorrent assisted streaming system for video-on-demand. In *Proc. IEEE MMSP*, 2005.

[8] D. Eager, E. Lazowska, and J. Zahorjan. Adaptive load sharing in homogeneous distributed systems. *IEEE Trans. software engineering*, SE-12(5):662–675, 1986.

[9] C. Gkantsidis and P. Rodriguez. Network coding for large scale content distribution. In *Proc. IEEE INFOCOM*, 2005.

[10] W. Hoeffding. *Masstabinvariante korrelationstheorie*. Schriften des Mathematischen Instituts und des Instituts fur Angewandte Mathematik der Universitat Berlin 5, 1940.

[11] International engineering consortium. Digital subscriber line access multiplexer (DSLAM): Definition and overview. *http://www.iec.org/online/tutorials/dslam*.

[12] F. Kelly. *Reversibility and stochastic networks*. Wiley, New York, 1979.

[13] L. Kleinrock. *Queueing systems Volume 1:Theory*. Wiley Interscience, 1975.

[14] J. Korst. Random duplicated assignment: An alternative striping in video servers. In *Proc. ACM Multimedia*, 1997.

[15] M. Luby. LT Codes. In *Proc. IEEE Symposium on foundation of computer science (FOCS)*. IEEE Computer Society, 2002.

[16] P. Maymounkov and D. Mazieres. Rateless codes and big downloads. In *Proc. the International Workshop on Peer-to-Peer Systems*, February 2003.

[17] R. Mirchandaney, D. Towsley, and J. Stankovic. Adaptive load sharing in heterogeneous distributed systems. *Journal of parallel and distributed computing*, 9:331–346, 1990.

[18] M. Mitzenmacher, A. Richa, and R. Sitaraman. The power of two random choices: A survey of the techniques and results. *Handbook of Randomized Computing*, 1:255–305, July 2001.

[19] http://www.pplive.com.

[20] A. Sharma, A. Bestavros, and I.Matta. dPAM: a distributed prefetching protocol for scalable asynchronous multicast in P2P systems. In *Proc. IEEE INFOCOM*, 2005.

[21] S.Sen, J. Rexford, and D. Towsley. Proxy prefix caching for multimedia streams. In *Proc. IEEE INFOCOM*, 1999.

[22] S. Tewari and L. Kleinrock. Analytical model for BitTorrent-based live video streaming. In *Proc. IEEE NIME Workshop*, 2007.

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

[24] N. Vvedenskaya, R. Dobrushin, and F. Karpelevich. Queueing system with selection of the shortest of two queues: an asymptotic approach. *Problems of information transmission*, 1996.

[25] W. Whitt. *Bivariate distributions with given marginal*. Annals of Statistics 4, 1976.

[26] X. Zhang, J. Liu, B. Li, and T. Yum. CoolStreaming/DONet: A data-driven overlay network for live media streaming. In *Proc. IEEE INFOCOM*, 2005.

9. APPENDIX

Appendix

PROOF. (of Theorem 1) The service time distribution is assumed to be a finite mixture of exponential distributions. Let π_i be the weight of the i-th component in the mixture, and μ_i the parameter of the i-th exponential distribution, $i = 1, \dots, I$. When it is clear from context that quantities depend on the system index ℓ, we omit the subscript ℓ to simplify notations.

We adopt the following description of the state space of the system: a job, when it starts service, determines its type $i \in \{1, \dots, I\}$ according to the probability distribution $\{\pi_i\}$. Its type specifies the parameter of its exponential service time distribution. We let X_i denote the number of type i jobs in service, and Y the number of jobs currently waiting. Thus one has the constraint that $\sum_{i=1}^I X_i \leq N$, and $Y = 0$ when $\sum_{i=1}^I X_i < N$ (jobs enter service as soon as a service slot is freed).

It is easily seen that $((X_i)_{i=1,\dots,I}, Y)$ is a Markov process. Let $q((X,Y),(X',Y'))$ denote the transition rate from state (X,Y) to state (X',Y') for this process. Also, denote by e_i the i-th unit vector in $\mathbb{R}^I$. It is then easily seen that the non-zero transition rates are specified by:

$$
\begin{array}{l}
q((X,Y),(X,Y+1)) = \nu \text{ if } \sum_{i=1}^I X_i = N, \\
q((X,0),(X+e_i,0)) = \nu \text{ if } \sum_{i=1}^I X_i < N, \\
q((X,Y),(X-e_i+e_j,Y-1)) = \mu_i X_i / N\pi_j \text{ if } Y > 0, \\
q((X,0),(X-e_i,0)) = \mu_i X_i / \sum_{j=1}^I X_j \text{ if } Y = 0.
\end{array}
$$

Define now the time- and space-rescaled processes, $x_i(t) = (1-\rho)X_i(t/(1-\rho))$ and $y(t) = (1-\rho)Y(t/(1-\rho))$. Using results of Kurtz, and the fact that $1-\rho \to 0$ as $\ell \to \infty$, it can be shown that for any time interval $[0,T]$, there is convergence in probability of the rescaled processes (x,y) towards the set of fluid trajectories defined below. Process convergence is for the uniform norm

$$
||(x,y)-(x',y')|| := \sup_{t\in[0,T]} \left[|y(t)-y'(t)| + \sum_{i=1}^I |x_i(t)-x_i'(t)| \right].
$$

To simplify notation, we note $s = \sum_{i=1}^I x_i$. The fluid trajectories in the above statement are defined as non-negative absolutely continuous functions $y(t)$, $x_i(t)$ such that for almost every $t \in [0,T]$,

$$
\begin{array}{ll}
\frac{d}{dt}y(t) & = \nu f(t) - \sum_{i=1}^I \mu_i \frac{x_i(t)}{s(t)} g(t), \\
\frac{d}{dt}x_i(t) & = \nu\pi_i(1-f(t)) + \pi_i \sum_{j=1}^I \mu_j \frac{x_j(t)}{s(t)} g(t) - \mu_i \frac{x_i(t)}{s(t)}.
\end{array}
$$

In the above, the functions f and g take their values in $[0,1]$, and are such that $f(t) = g(t) = 1$ whenever $y(t) > 0$, and $f(t) = g(t) = 0$ whenever $s(t) < m$. Also, the value for ν in the first line is the limiting value, that is

$$
\nu := \lim_{\ell\to\infty} \nu_\ell = \frac{1}{\sum_{i=1}^I \pi_i/\mu_i}\,. \tag{20}
$$

This is just rewriting that the load reaches one in the $\ell \to \infty$ limit.

Using the fact that $y(t) \geq 0$, it follows from the first equation that when $y(t) = 0$, necessarily

$$
\sum_{i=1}^I \mu_i \frac{x_i(t)}{s(t)} g(t) = \nu f(t).
$$

Thus by setting $h(t) = 1$ if $y(t) > 0$ or $y(t) = 0$ and $\sum_{i=1}^I x_i(t) = n$, and $h(t) = 0$ otherwise, one thus obtains

$$
\frac{d}{dt}x_i(t) = \pi_i \left[h(t)\nu + (1-h(t)) \sum_{j=1}^I \mu_j \frac{x_j(t)}{s(t)} \right] - \mu_i \frac{x_i(t)}{s(t)}.
$$

We now establish the following

LEMMA 2. *Let*

$$
q_i := \frac{\pi_i/\mu_i}{\sum_{j=1}^I \pi_j/\mu_j}, \quad j = 1, \dots, I,
$$

and introduce the function

$$
L(x) := \sum_{i=1}^I x_i \log\left(\frac{x_i}{s} \frac{1}{q_i} \right).
$$

Note that, up to a factor $s = \sum_{i=1}^I x_i$, this coincides with the Kullback-Leibler divergence between the distributions $\{x_i/s\}$ and $\{\pi_i\}$. Then the function $L(x(t))$ is strictly decreasing along a fluid trajectory $x(t)$, except when

$$
\frac{x_i}{s} = q_i,\ i = 1, \dots, I.
$$

PROOF. Introduce the notation $s = \sum_{i=1}^I x_i$. Write

$$
\begin{array}{l}
\frac{d}{dt}L(x) = \sum_{i=1}^I \frac{d}{dt}x_i \left[\log(x_i) - \log(s) - \log(q_i) \right] \\
= \frac{h}{s} \sum_{i=1} \left[\pi_i \nu s - \mu_i x_i \right] \left[\log(x_i) - \log(s) - \log(q_i) \right] \\
+ \frac{1-h}{s} \sum_{i=1} \left[\pi_i \sum_{j=1}^I \mu_j x_j - \mu_i x_i \right] \left[\log(x_i) - \log(s) - \log(q_i) \right].
\end{array}
$$

We show that both terms in the right-hand side are non-negative. Using expression (20), the first term reads

$$
\begin{array}{l}
h \sum_{i=1}^I \mu_i \left[\frac{\pi_i/\mu_i}{\sum_{j=1}^I \pi_j/\mu_j} - \frac{x_i}{s} \right] \left[\log(x_i/s) - \log(q_i) \right] \\
= h \sum_{i=1}^I \mu_i \left[q_i - \frac{x_i}{s} \right] \left[\log(x_i/s) - \log(q_i) \right].
\end{array}
$$

It is clear in this expression that each term in the sum is non-positive, and the sum equals zero only if the distribution (x_i/s) coincides with the distribution (q_i).

The argument for the second term goes as follows. One has to establish the following inequality:

$$
\sum_{i=1}^I \pi_i \sum_{j=1}^I \mu_j x_j \log(x_i/(sq_i)) \leq \sum_{i=1}^I \mu_i x_i \log(x_i/(sq_i)), \tag{21}
$$

and that equality holds in the above only if $x_i/s \equiv q_i$. By definition of q_i, it holds that

$$
\pi_i = \frac{q_i\mu_i}{\sum_{j=1}^I q_j\mu_j}.
$$

Introduce the notation $u_j = x_j/(sq_j)$. After division of both sides of (21) by $s\sum_{j=1}^I \mu_j q_j$, this reads, using the previous expression for π_i,

$$
\sum_{j=1}^I \pi_j u_j \sum_{i=1}^I \pi_i \log(u_i) \leq \sum_{i=1}^I \pi_i u_i \log(u_i). \tag{22}
$$

An inequality due to Hoeffding [10] (see also [25] and [4] for more easily accessible references) states that, given two random variables U, V with identical distributions, for any two non-decreasing functions $f, g : \mathbb{R} \to \mathbb{R}$ such that $f(U)$ and $g(U)$ have finite variances, one has:

$$
\mathbf{E}[f(U)g(V)] \leq \mathbf{E}[f(U)g(U)].
$$

Note that the inequality (22) we need to prove is of that form, with as non-decreasing functions $f(U) = U$ and $g(U) = \log(U)$. Finiteness of variances is trivially satisfied as the random variables U take only finitely many values. This establishes the desired inequality (22), and the equivalent inequality (21).

Equality in (21) implies equality in (22). However, the latter holds if and only the distributions of $(f(U), g(V))$ and $(f(U), g(U))$ coincide; see e.g. [25]. As the functions f, g are strictly increasing, this in turn holds if the distributions of (U, V) and (U, U) coincide. In the present case, this holds only if $u_i = x_i/(sq_i)$ is constant. This concludes the proof of the Lemma. □

The lemma has the following consequence, the proof of which we omit here.

COROLLARY 1. *Consider any subsequence ℓ' converging to infinity, and such that the distribution of the rescaled state vector $(1-\rho_\ell)(X^\ell, Y_\ell)$ of the ℓ'-th system in steady state converges weakly to some limiting distribution as $\ell' \to \infty$. Then under the limiting distribution, with probability 1 it holds that*

$$\frac{X_i}{\sum_{j=1}^I X_j} = q_i,\ i = 1, \dots, I.$$

The weak convergence result (7) is then established as follows. The Pollaczek-Khinchin formula (see e.g. [3], p.183, eq. (4.3.4)) characterises the Laplace transform of the workload V^ℓ in the ℓ-th system in steady state as

$$\mathbf{E}[e^{-\alpha V^\ell}] = \frac{\alpha(1-\rho_\ell)}{\alpha + \nu_\ell[\mathbf{E}(e^{iu\sigma} - 1)]}.$$

Replacing α by $(1-\rho_\ell)\alpha$ in the above, and letting ℓ tend to infinity, it readily follows that

$$\lim_{\ell\to\infty} \mathbf{E}[e^{-\alpha(1-\rho_\ell)V^\ell}] = \frac{1}{1 + \nu_\infty \alpha \overline{\sigma^2}/2}$$

Using the fact that $\nu_\infty \overline{\sigma} = 1$, we can readily identify the right-hand side as the Laplace transform of an Exponential random variable with mean $\overline{\sigma^2}/2\overline{\sigma}$.

Noting that the workload V^ℓ can be written as

$$V^\ell = \sum_{n=1}^{Y^\ell} \sigma_n + \sum_{i=1}^I \sum_{n=1}^{X_i^\ell} \sigma_{i,n},$$

where σ_n are i.i.d. service times drawn from the original service time distribution, and $\sigma_{i,n}$ are i.i.d. exponential random variables with parameter μ_i, weak convergence of the rescaled workload $(1-\rho_\ell)V^\ell$ entails that the sequence of rescaled state variables $(1-\rho_\ell)(X^\ell, Y_\ell)$ is tight. It further entails that, for any limiting distribution, one has

$$\mathbf{P}(\overline{\sigma}Y + \sum_{i=1}^I \frac{1}{\mu_i} X_i > t) = e^{-2t\overline{\sigma}/\overline{\sigma^2}}.$$

Combined with the result of the corollary, this guarantees weak convergence of $(1-\rho_\ell)(X^\ell, Y_\ell)$ to a distribution which is such that with probability 1, $X_i/S = q_i$, where $S = \sum_{j=1}^I X_j$, $S \le m$, $S < m$ implies $Y = 0$, and finally

$$\mathbf{P}(\overline{\sigma}Y + S\sum_{i=1}^I \frac{q_i}{\mu_i} > t) = e^{-2t\overline{\sigma}/\overline{\sigma^2}}$$

Note now that

$$\overline{\sigma^2} = 2\sum_{i=1}^I \pi_i/(\mu_i^2).$$

The above equation can then be rewritten

$$\mathbf{P}(\overline{\sigma}Y + S\overline{\sigma^2}/(2\overline{\sigma}) > t) = e^{-2t\overline{\sigma}/\overline{\sigma^2}}$$

Take now $x > m$. One then has

$$\mathbf{P}(S + Y > m) = \mathbf{P}(\overline{\sigma}Y + S\overline{\sigma^2}/(2\overline{\sigma}) > t),$$

with $t = m\overline{\sigma^2}/2 + \overline{\sigma}(x-m)$. Plugging this value for t in the right-hand side of the previous identity yields the first half of (7). The second half follows from a similar argument.

The proof of (8) goes as follows. When the service box is full, the departure rate from the service box is exactly

$$\sum_{i=1}^I \mu_i q_i = \frac{1}{\overline{\sigma}}.$$

One can thus map the probability of having a rescaled waiting time larger than t to the probability of having a rescaled number of jobs already waiting upon arrival of order $t/\overline{\sigma}$. Thus,

$$\begin{aligned}\lim_{\ell\to\infty} \mathbf{P}((1-\rho_\ell)W^\ell > t) &= \mathbf{P}(\overline{\sigma}Y + S\overline{\sigma^2}/(2\overline{\sigma}) > t + m\overline{\sigma^2}/(2\overline{\sigma}) \\ &= e^{-m - 2t\overline{\sigma}/\overline{\sigma^2}}.\end{aligned}$$

Finally, to establish (9), write

$$\begin{aligned}\lim_{\ell\to\infty} \mathbf{P}(W^\ell = 0) &= \lim_{\ell\to\infty} \mathbf{P}(Z^\ell < N^\ell) \\ &= 1 - e^{-m}\end{aligned}$$

in view of (7).