M Ű E G Y E T E M 1 7 8 2

Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics

Performance modelling and analysis of distributed P2P download systems

Juhos Attila

Supervisor:
Dr. Levendovszky János
Department of Networked Systems and Services

October 26, 2017

Contents

1 Abstract

The objective of the paper is to develop a stochastic model for P2P download processes and investigate the performance and scalability of such a system. Based on this analysis and the corresponding simulation results, one can evaluate the necessary server capacity as well as the proper cache size to provide a given population of P2P users with fast download services.

In P2P systems peers negotiate to download parts (packets) of a desired file from other peers that already possess those packets. A peer will only turn to the server for a packet if it cannot be downloaded from the other peers. An inappropriate design of the system, in terms of ill-founded download- and packet storing mechanisms, may easily result in a heavily overloaded server that will ultimately refuse to serve the demands of the clients, leading to system overflow and break-down of QoS. Nonetheless, a well-designed system minimises the load of the server enabling efficient system scaling.

The peer birth rate of P2P download systems is assumed to be of Poissonian nature characterised by a single parameter, λ. This parameter along with the applied download and cache strategies will determine the minimum server capacity (C) required to serve the peers. Up to the present date, however, only few studies have been conducted in calculating the minimum capacity needed by a server to maintain a flawless download process. Furthermore, these analyses are mostly based on oversimplified models and can lead to ill-founded conclusions. As a result, in this paper, a general stochastic download model is introduced, in which framework the relationship between the minimum required server capacity, the cache size and download strategy are sought.

The results are treated in the following structure:

a) At first, a system model was created, from which we determined a recursive formula calculating the download state of peers. The state transition depends on the birth rate of peers and on the cache strategy.

b) A complex simulation package was produced in order to evaluate the system performance numerically in case of different peer birth rate protocols and cache strategies. The software calculates the distribution of clients in the system for the entire time of interest. Besides that, we could plot and analyse the entropy of peer distribution in different download states, which provides a major quality characteristic of the system.

Based on this model, we derived the state transitions of the system as well as the average progress of download, which proves to be an important quality parameter of the system.

The numerical experiments have demonstrated that the system performance depends greatly on the cache size and strategy as well as on the server capacity, i.e.

1. in the case of deterministic cache strategy, one is prompted to provide a fairly large server capacity (e.g. multiple times the mean of the peer birth rate distribution).

2. or design a cache strategy that focuses on retaining those packets that are the most scarcely possessed by the set of peers.

The findings of the paper may offer insight into the optimisation of cache strategies for real-world download systems. An elaborate design may help to implement a system that remains robust against massive user populations. As far as the future work is concerned, the model may as well be extended to the use of network coding technologies. Any future research in the field of efficiently scaling P2P download systems will potentially lead to the global application of this technology.

2 Introduction

From the early onset of networking, designers were well aware that the current network configuration and the capacity of server environments do not facilitate the transfer of large data files over the internet to multiple users. Soon peer-to-peer (P2P) file exchanging protocols arose to support large data traffic. The concept behind these protocols resides in the utilisation of end user upload capacity. With the guidance of the central server, users can exchange data without interfering with the server any more, thus taking over the burden of overloaded servers. This development has proven to be instrumental to present day networking as well, when mobile (e.g. 5G) networking is aiming at better server utilisation and energy awareness. Thus P2P file downloads can decrease server load and utilize network resources more efficiently.

Many file-sharing systems have already been developed, but BitTorrent has proven to outperform most of them. It is worth recalling that early peer-to-peer systems supported users (*peers*) uploading their content only as soon as the file was downloaded in its entirety. The disadvantage of this approach is that peers tend to leave the system as soon as the download has been finished, thus the download process could not be maintained easily. In contrast, BitTorrent-protocol shares files fragmented into smaller chunks (which are usually called *packets, fragments, pieces*) and the download is taking place on a packet-by-packet basis. The advantage of this mechanism is that peers may as well start sharing the received content no matter how advanced they are in the downloading process.

In spite of BitTorrent being a more or less mature technology, there is still room for development of specialised applications of P2P protocols. There have been recent attempts to introduce peer-to-peer protocols into live streaming services, an environment fairly different from the usual one, where the file of interest is totally available. [1] outlines the importance of these recent ideas, since live streaming data traffic is becoming more and more prevailing. For instance, sport events often gather so many online spectators that the system is responsible for serving hundreds of thousands of users. According to [4], internet video traffic represented more than the half of total internet traffic in 2016 and this percentage further increases, reaching 67.64% by 2021 (depicted in figure 1).

Therefore, P2P system designers are in an urgent need for developing a more realistic model of such a system in order to determine the basic design parameters, such as the server capacity and (in case of a live streaming context) the optimal cache strategy as well as the forward windows size. These parameters are crucial for well-performing download systems, since the full

[1]Source: [4]

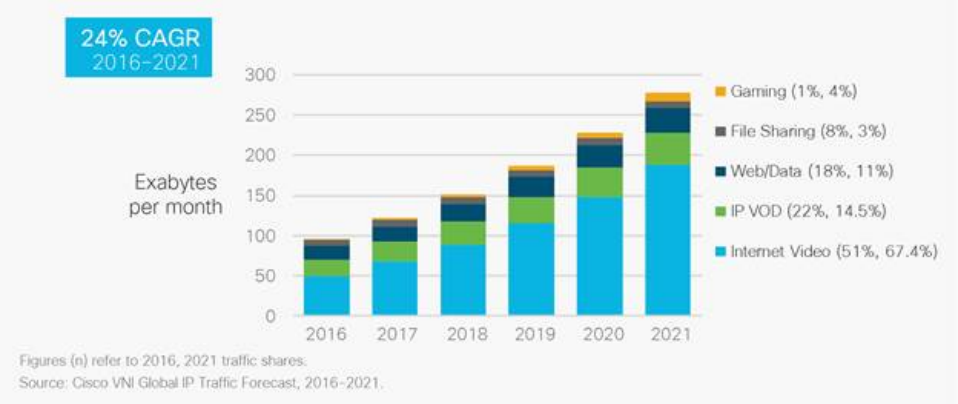

Figure 1: Internet traffic from 2016 till 2021 foreseen by Cisco [1]

range of client needs (e.g. continuous video quality with no interrupts, relatively low latency) must be addressed when the scale of user population exceeds bounds no less than hundreds of thousands. [3] reports the example of a PPLive network ([5]) which was found to have kept connection with over 100.000 clients. Moreover, besides facing serious system design problems, service providers are also challenged by the competitive nature of the market. [1] points out the fluctuation of peers to other providers whenever large deviations from a flawless service are encountered.

The key objective of this work is to provide designers with an analytical model of P2P systems that is more precise and more plausible than the outcome of previous attempts. Before commencing with the discussion, the major issues of a BitTorrent session are outlined. [2] focuses on some key features, two of which are the scalability, i.e. the handling of massive user population and efficiency, namely at what pace peers download the desired content and how well their bandwidths are utilised.

The rest of the paper is organised as follows. In section 3, a brief description of the BitTorrent protocol is discussed. We refer to the background knowledge and related work, namely we are going to outline previous results on the importance of peer group size and number of fragments. Besides, the *warming up* period of P2P systems is examined. Section 4 introduces our approach and our stochastic model of P2P systems. The key contribution of our work is the generality provided by our model since parameters like the applied cache size or the cache strategy are changeable and experimenting with various peer birth models is possible, as well. Section 5 outlines the numerical results achieved by the previously designed model.

3 Background and Related Work

In this section, existing P2P systems are briefly summarised and compared to our work according to papers [1, 2].

One of the implementations of P2P systems is the BitTorrent ([6]) protocol. The basic unit of service in BitTorrent is called *session*, which consists of a *file* and a dynamic set of clients/users, the so-called *peers* that aim at downloading the file, the desired content. The large set of peer formation is usually called a *swarm*, which is populated by *leechers* (peers in a session that are still downloading) and *seeders* (peers in a session that finished downloading but still offer their upload bandwidth so that other clients could receive the file). In order to join a session, peers must obtain a *torrent file* that includes the list of *fragments* in which the file was broken. Peers also receive a list of participant peers to which they can build and maintain communication links. This is the first stage, when newly born peers can indeed start downloading effective content.

Peers send download requests to some of their connections. Although peers maintain connection to 40 to 100 peers (depending on implementation and the actual size of the swarm), they only upload content to a handful of peers, due to limited upload bandwidth. This means that not all of the download requests will be accepted, they need to be filtered. Two major upload-download policies are applied in BitTorrent-like systems. First of all, peers solicit fragments that are the least available among other peers. This policy is often referred to as the *rarest-first-download* policy ([1]). Secondly, peers bombarded with multiple concurrent download requests prefer uploading data to peers that prove to be uploading content at the fastest rate. This tactic is named *Tit-for-Tat* upload policy. This rule of thumb is considered by [1] and others([7]) the reason of BitTorrent's overwhelming and general success.

Much work has also been done in the field of *optimistic unchoking*, a mechanism to select unknown peers for download. Although this field is not connected closely to our topic, it gives an insight into how the dynamics of BitTorrent systems works. Tit-for-Tat policy discriminates most of the peers that they have link to, without periodically measuring their upload bandwidth, which might have changed meanwhile. Moreover, freshly arrived peers are underscored due to their incapability to upload data, no matter how rich they are in upload capacity. To solve this issue, peers regularly run optimistic unchoking algorithms to get some of these isolated peers involved into the system. The pure purpose of this algorithm is to randomly select a peer for bidirectional data transfer. The selection is considered random, since little is known about the upload capacity or willingness of the chosen peer. This selection is periodically run, every 30 seconds([2]). The drawback of such a policy is the abuse of the system's cooperative nature

by malicious peers, often called *free-riders*. Free-riders are only concerned about acquiring the file without contributing their resources to the system by making their upload bandwidth unavailable. As demonstrated in [9], free-riders can achieve the fifth of a normal peer download rate by making use of optimistic unchoking. [8] suggests a new algorithm to handle such peers more efficiently. It basically maintains information about previously unchoked peers and their contribution to the network, this way favouring good peers or ignoring free-riders.

From the previous arguments one can deduce that there are two basic static models of P2P systems. The first of them is proposed by [2] and it divides the user population into several little groups, which exchange data exclusively among themselves. The second model, applied by [1], commences from a one-cluster model, where peers participate in a connected network. Since the latter makes possible to deduce information about the multi-group model, we are going to adopt the one-cluster model in the forthcoming sections.

It is also a key feature of BitTorrent-like systems, that they split the data file into numerous fragments (or *packets*). The number of these packets is dependent on the context in which the system is used. Systems designed to share large data files that are available before the P2P session split the file into a few thousand pieces, while live streaming services split the actually available data (a few seconds, at most a few minutes of video streaming) only into a few dozens or even less packets. The argument for fragmentation of the file is (as outlined in previous sections) that peers may start acting like data sources as soon as the first packet has been downloaded, and waiting for the file to be received in its entirety is not necessary any more. This way even users are being prompted to contribute to a sustainable session by allowing a fraction of their upload capacity to be used, since they will still be downloading when upload starts. The reason why further increasing the number of pieces is impeded is that this process would involve reducing the size of individual packets. The smaller the packet is, the greater the proportion of network headers are when sending the packets through the internet network. Since upload capacities of users are tightly limited, this proportion must be kept low.

3.1 Peer group size and number of packets available for download

The previous paragraph outlined some important parameters of BitTorrent sessions, namely the *peer group size*, the *number of packets available for download*, the *fraction of utilisation of end-user capacity*. Up to the present day, many surveys have been conducted on determining the right choice for these parameters, but some of the results prove to be controversial. For

example, the peer group size is proven to have very low impact on the efficiency of the download process by [1]. The authors of [1] defined the *efficiency of the file exchange* (i.e. the fraction of the total peer upload capacity that can be utilized) as the probability that in a group of k peers there exists at least one peer that needs one of the fragments which another peer is has, denoted by η. They computed this value to be:

$$\eta = 1 - \sum_{n_i=0}^{N-1} \frac{1}{N} \left(\frac{N - n_i}{N(n_i + 1)} \right)^k$$

where N denotes the number of fragments available for download and k is the number of peers participating in the file-exchange. Expanding this equation it can be written that:

$$\eta = 1 - \frac{1}{N} - \frac{R}{N}$$

where

$$R = \frac{1}{2^k} \left(1 - \frac{1}{N}\right)^k + \frac{1}{3^k} \left(1 - \frac{2}{N}\right)^k + \ldots + \frac{1}{(N-1)^k} \left(\frac{2}{N}\right)^k + \frac{1}{N^k} \left(\frac{1}{N}\right)^k$$

With sufficiently large peer groups ($k > 10$) R/N becomes negligible compared to $1/N$ and, therefore, the efficiency can be approximated as:

$$\eta \approx 1 - \frac{1}{N}$$

Thus paper [1] could prove that the efficiency of file exchange does not depend on the peer group size on the account of there being at least around 10 clients. Moreover, the dependency on the number of fragments could also be pointed out. With a fairly large number of packets available, almost the entire end-user upload capacity could be utilised. Figure 2 plots the effect of the peer group size on the efficiency of file sharing with different values of the number of fragments available for downloading. The set of curves underlines the previous conclusions.

3.2 BitTorrent warming up period

Finally, it is worth mentioning the *warming up* period reported by BitTorrent-users mentioned in [10] and [2]. C. Barakat and I. Pratt ([10]) conducted an experiment on the amount of data received and uploaded during a P2P session with an instrumented client behind a 10Mb/s campus network access link. The results are outlined in figure 3. It is obvious that the derivative

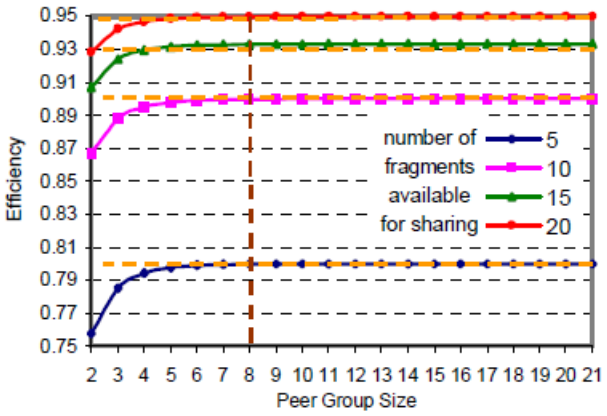

Figure 2: Variation of the efficiency of file sharing for different quantities of fragments available for exchanging [2]

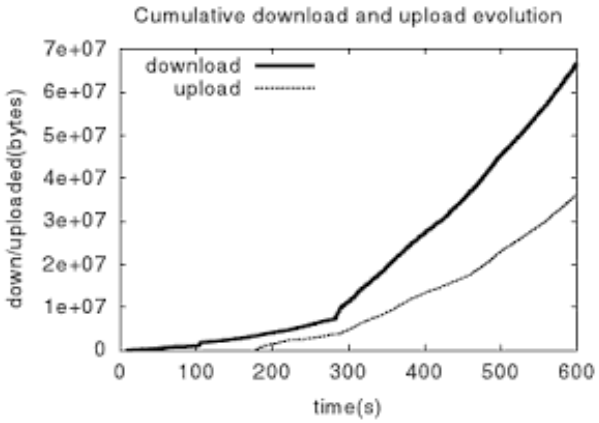

Figure 3: Download and upload evolution during a session through a 10Mb/s link. [3]

of the curve not being constant but increasing, there is a warm up period preceding the steady state.

In their article, Y. Yue et al. provide a so-called *fluid* model of P2P sessions. In their paper the use of a multi-group model is suggested, where leechers form several little sets without intergroup data exchange and seeders form a single group maintaining contact with each other. The authors analyse the performance of the system by using a continuous time parameter. Their main objective is to find the parameters of the system in steady states and to determine the convergence time of peers to their steady state. This convergence time is mainly responsible for the aforementioned warming-up period, when users face low download rates. After examining the unchoking algorithm used by BitTorrent the explanation is straightforward: until the given peer can obtain enough data to start uploading, no one will choose to keep contact with them even after optimistic unchoking. [2] presents a meticulous mathematical model to prove this.

Let us suppose that the swarm consists of N peers shared between n peer groups: G_i with the number of participants N_i, where $i = \overline{1,n}$. The main question is how long it takes for a peer to reach its steady state. In order to simplify the question, we are going to work in case of $n = 2$ with group G_1 consisting of peers with high upload capacity and G_2 consisting of peers with low upload bandwidth. Let us suppose that peers upload data only to n_u other peers (this number is usually around 3 or 4). A peer reaches its steady state as soon as it finds n_u peers with the help of optimistic unchoking, since regular unchoking cannot be used due to the lack of network information. Since optimistic unchoking can be applied every 30s, the question is reduced to the number of rounds necessary to reach the steady state. Let $\mathcal{R}$ denote the random variable corresponding to the number of optimistic unchoking rounds necessary for a peer to build up a link to n_u different peers. The probability of finishing after exactly the k^{th} round is:[4]

$$P_{n_u}(\mathcal{R} = k) = P_{n_u-1}(\mathcal{R} \leq k-1) \cdot P(\text{leecher finds a } G_1 \text{ leecher in round } k)$$

This can be calculated as following:

$$P_{n_u}(\mathcal{R} = k) = \frac{\binom{N_1-1}{n_u-1}\binom{N_2}{(k-1)-(n_u-1)}}{\binom{N_1+N_2-1}{k-1}} \cdot \frac{N_1 - n_u}{N_1 + N_2 - k}$$

After transforming the formula and calculating the mean of $\mathcal{R}$ Yue et al. found that:

$$\mathbb{E}(\mathcal{R}) = n_u \binom{N_1 - 1}{n_u} \sum_{k=n_u}^{N-N_1+n_u} \frac{\binom{N-N_1}{k-n_u}}{\binom{N-1}{k}}$$

[2]Source: [1]

[3]Source: [10]

[4]For a more detailed explanation see [2]

According to the established formula, the article plotted the relationship between $\mathbb{E}(\mathcal{R})$ and G_1's portion among all leechers in figure 4. The figure shows that the convergence time decreases fast with the portion of G_1 increasing. For example, with a proportion of 10% the expected number of rounds is around 20, i.e. the given peer would reach constant state after around 10 minutes.

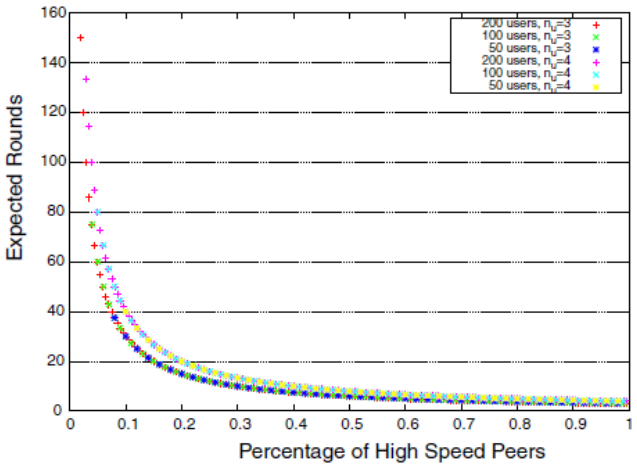

Figure 4: Expected number of rounds after which a peer in group $G1$ reaches a steady state in function of G_1's portion among all leechers. [5]

In conclusion, up till now a massive effort and research have been invested in the investigation of P2P protocols, but there are still some open questions. These questions are especially important in the advent of 5G mobile technology, where a large number of users are supposed to have access to a large amount of data with low latency and at high data speed. On the one hand, the model proposed in [1] can only handle a system in steady state without taking serious transient phenomena into consideration, like the initial formation of the swarm, the gradual convergence of peers to a steady download state or the way seeders abandon the session. On the other hand, the model suggested by [2] introduces a reasonable insight into the calculation of convergence time, but it lacks the simulation of inter-steady states. The key objective of our work is to provide a stochastic model of BitTorrent-like systems and to offer a method for performance analysis in order to address these questions, too. Previous papers in the field would usually try to estimate the performance in peak situations, characterised as steady states. The argument against this approach is that many times more massive performance is needed in case of huge system changes, triggered by a large peer growth, rather than situations with large peer population. Informally speaking, points with large deviation, large gradient are just as

[5]Source: [2]

important to discuss as points with "zero" gradient. Our work aims at promoting a model where such changes are handled analytically. The objective is also to provide numerical results on the minimum required server capacity that will most likely cope with massive user populations subject to massive deviations in peer birth rate. Later we will realise that this problem does not lend itself to straightforward analysis because of the random peer births.

4 General stochastic model of a P2P system

In this chapter we introduce the general stochastic model for P2P systems. Our approach provides information about the general dynamics of the system described by the number of peers in different states of download. In this case the dynamics of the individual peer download process are of marginal importance and the main goal is to provide a well-performing design that will stay robust in the case of massive client populations.

Let us assume that we have a single user cluster (the results obtained for a single cluster can easily be generalized to multi-clusters), and that the state of the system evolves on a discrete time scale, where the time variable is denoted by k. The file to be received is denoted by $\mathcal{F}$ and consists of L packets with the same size, and it is cut into M blocks, where each block is constructed from K packets. In our work we introduce this smaller but not packet-level data unit, the block, to provide a segmentation of the original file, where the segments must be downloaded consecutively, i.e. in case of videos in the sequence of video snippets. The need for this data unit is to provide a model for live streaming applications as well, where it is crucial to download packets within a given latency. The larger a block (the bigger K) is, the more fragments are available for download for a certain peer, but then there is an increased latency.

Since blocks ought to be received in their predefined order (even though packets within a block may still be downloaded in a random sequence), the state of each peer can be characterized by an index i indicating the last block it managed to download. Given that $1 \leq i < M$, a peer found in state i is competing for packets from block $i + 1$. Peers in state M have downloaded the entire file, they are only acting as seeders to support the flow of the system. Newly born peers are automatically found in state 1 (this could correspond to the fact that they have recently received the *torrentfile* from the server).

It is important to note that clients tend to emerge according to some random distributions. It is often assumed that population models are of Poissonian nature. Nonetheless, we are going to handle a more general model, in which peers emerge in accordance with a random variable $\mathscr{A}$. Besides that, it is presumed that peers in state M tend to remain present

in the system with a probability of P_{stay}. Once a seeder has left the system, it never returns.

The number of peers in a given state $i \in \{1, 2, \ldots, M\}$, at time instant k is denoted by $n_i(k)$. Therefore, the system is characterized by a state vector $\underline{n}(k) \in \mathbb{N}^M$, $\underline{n}(k) = (n_1(k), n_2(k), \ldots, n_M(k))$.

The server capacity is denoted by C. This capacity shows how many download requests can be handled purely by the server at each time instant. Besides that, each of the peers in the swarm possess the same amount of cache and use it with the same download storage mechanism, which we are going to refer to as *cache strategy*. Different cache strategies are discussed later.

Consequently, an S file sharing scenario consists of a tuple of a file, a server capacity, a cache strategy, a peer birth random variable and a probability of a seeder remaining active for another interval:

$$S =< \mathcal{F}(M, K), C, cache\ strategy, \mathscr{A}, P_{stay} > \tag{4.1}$$

Such a file sharing scenario can produce different $\underline{n}(k)$ state vector evolution due to the random peer birth rate. The set of different dynamics is denoted by

$$\mathcal{N}_S = \{\underline{n}(k) \text{ state vector function result of file sharing scenario } S\}$$

Our objective is to analytically describe the state transition rule of $\underline{n}(k)$. It is assumed that a time interval lasts long enough for peers being able to download the entire information from a single block. Note that packets are still received in a random order within a block. However, this assumption does not restrict our model in any way, since blocks are to be downloaded in the right order.

At first, let $\xi_{i,j}(k) \in \{0, 1\}$ denote whether a peer j found in state i at time instant k is not able to download the information from block $i+1$ from another peer. Secondly, let $p_i(k)$ denote the probability that a certain peer in state i is able to receive the information from block $i + 1$ purely from other peers. Consequently, it can be written that

$$p_i(k) = P(\xi_{i,j}(k) = 0) \text{ and } \overline{p_i}(k) = 1 - p_i(k) = P(\xi_{i,j}(k) = 1)$$

Then, let $\tilde{p}(k)$ denote the probability, that a peer at time instant k can download the relevant information directly from the server. Since the server has capacity C, one can write that:

$$\tilde{p}(k) = P(\sum_{i=1}^{M-1} \sum_{j=1}^{n_i(k)} \xi_{i,j}(k) \leq C) \tag{4.2}$$

Note that the first summation runs up to index $M-1$, because peers in state M will not need downloading packets any more. In order to expand this probability, we will use the Chernoff bound, which states that given a random variable Y, then

$$P(Y > C) = P(e^{sY} > e^{sC}) < \frac{\mathbb{E}(e^{sY})}{e^{sC}}$$

holds for any $s > 0$. We apply this inequality for our case. Let $Y = \sum_{i=1}^{M-1} \sum_{j=1}^{n_i(k)} \xi_{i,j}(k)$ and let C denote the server capacity. We find that, that for any $s > 0$:

$$P(\sum_{i=1}^{M-1} \sum_{j=1}^{n_i(k)} \xi_{i,j}(k) > C) < \frac{\mathbb{E}\left[\exp(s \sum_{i=1}^{M-1} \sum_{j=1}^{n_i(k)} \xi_{i,j}(k))\right]}{e^{sC}} \tag{4.3}$$

Now we expand on the numerator:

$$\mathbb{E}\left[\exp(s \sum_{i=1}^{M-1} \sum_{j=1}^{n_i(k)} \xi_{i,j}(k))\right] = \mathbb{E}\left[\prod_{i=1}^{M-1} \prod_{j=1}^{n_i(k)} \exp(s\xi_{i,j}(k))\right]$$

We now assume that the random variables $\xi_{i,j}(k)$ are mutually independent and it can be further deduced that:

$$\begin{aligned}
\mathbb{E}\left[\exp(s \sum_{i=1}^{M-1} \sum_{j=1}^{n_i(k)} \xi_{i,j}(k))\right] &= \prod_{i=1}^{M-1} \prod_{j=1}^{n_i(k)} \mathbb{E}\left[\exp(s\xi_{i,j}(k))\right] \\
&= \prod_{i=1}^{M-1} \prod_{j=1}^{n_i(k)} (p_i(k)e^{s\cdot 0} + (1-p_i(k))e^{s\cdot 1}) \\
&= \prod_{i=1}^{M-1} \prod_{j=1}^{n_i(k)} (p_i(k) + (1-p_i(k))e^{s})
\end{aligned}$$

We introduce the following notation:

$$\mu_i^{(k)}(s) = \ln(p_i(k) + (1-p_i(k))e^s), \tag{4.4}$$

where $\mu_i^{(k)}(s)$ is called the logarithmic generator function of a peer in state i at time instant k. Thus, it can be written that:

$$\begin{aligned}
\mathbb{E}\left[\exp(s \sum_{i=1}^{M-1} \sum_{j=1}^{n_i(k)} \xi_{i,j}(k))\right] &= \prod_{i=1}^{M-1} \prod_{j=1}^{n_i(k)} e^{\mu_i^{(k)}(s)} \\
&= \exp\left[\sum_{i=1}^{M-1} \sum_{j=1}^{n_i(k)} \mu_i^{(k)}(s)\right]
\end{aligned}$$

Returning to equation (4.3), we found that:

$$P(\sum_{i=1}^{M-1}\sum_{j=1}^{n_i(k)}\xi_{i,j}(k) > C) < \frac{\exp\left[\sum_{i=1}^{M-1}\sum_{j=1}^{n_i(k)}\mu_i^{(k)}(s)\right]}{e^{sC}},$$

from where finally:

$$P(\sum_{i=1}^{M-1}\sum_{j=1}^{n_i(k)}\xi_{i,j}(k) > C) < \exp\left[-sC + \sum_{i=1}^{M-1}\sum_{j=1}^{n_i(k)}\mu_i^{(k)}(s)\right] \tag{4.5}$$

holds for any $s > 0$ numbers. Consequently, it follows that

$$P(\sum_{i=1}^{M-1}\sum_{j=1}^{n_i(k)}\xi_{i,j}(k) > C) \leq \min\left\{1, \min_{s>0}\left\{\exp\left[-sC + \sum_{i=1}^{M-1}\sum_{j=1}^{n_i(k)}\mu_i^{(k)}(s)\right]\right\}\right\}$$

This way the final formula of $\tilde{p}(k)$ can be estimated from below as:

$$\tilde{p}(k) > 1 - \min\left\{1, \min_{s>0}\left\{\exp\left[-sC + \sum_{i=1}^{M-1}\sum_{j=1}^{n_i(k)}\mu_i^{(k)}(s)\right]\right\}\right\} \tag{4.6}$$

Since the logarithmic generator function (4.4) does not depend on index j of a peer in state i (note that index j only represents the fact that there may be multiple concurrent users in state i), $\sum_{j=1}^{n_i(k)}\mu_i^{(k)}(s) = n_i(k)\cdot\mu_i^{(k)}(s)$.

Furthermore, since our purpose is to minimise the necessary server capacity, it is enough to estimate $\tilde{p}(k)$ with its lower estimation. In that case we will estimate the minimal server capacity from above, which will not lead to a under-performing design. Thus,

$$\tilde{p}(k) \approx 1 - \min\left\{1, \min_{s>0}\left\{\exp\underbrace{\left[-sC + \sum_{i=1}^{M-1} n_i(k)\cdot\mu_i^{(k)}(s)\right]}_{\Gamma(s)}\right\}\right\} \tag{4.7}$$

The argument of the inner exp function is to be denoted by $\Gamma(s)$, and we are going to call it the logarithmic generator function of the server upload. In order to calculate the minimum of $\Gamma(s)$, when $s > 0$, we adopted the Barzilai-Borwein method.

Corollary 1 *Let us have a convex, derivable function $F : \mathbb{R}^m \to \mathbb{R}$ with one single minimum point and with ∇F Lipschitz-function and the vectors $x_0 \neq$*

x_1 from $\mathbb{R}^m$. Let us construct the $\{\gamma_n\}, n \geq 1$ and $\{x_n\}, n \geq 2$ sequences of vectors from $\mathbb{R}^m$ in a such a way that:

$$\gamma_n = \frac{(x_n - x_{n-1})^T \left[\nabla F(x_n) - \nabla F(x_{n-1})\right]}{||\nabla F(x_n) - \nabla F(x_{n-1})||^2}, \qquad n \geq 1$$

$$x_{n+1} = x_n - \gamma_n \cdot \nabla F(x_n) \qquad n \geq 2$$

In this case the number sequence $\{x_n\}$ is convergent to the minimum point of F:

$$\lim_{n\to\infty} x_n = \arg\min_{x\in\mathbb{R}^m} F(x) \tag{4.8}$$

Without entering into details, it can be easily shown that our function $\Gamma(s)$ satisfies all the conditions of the corollary. In our simulation package we modified the algorithm suggested by the theorem in such a way that it would handle only s positive parameters.

The simplified protocol that is modelled now determines the state transition as follows: a certain peer in state i first attempts to download the required information (i.e. information from block $i+1$) from a fellow peer. The probability for this was denoted by $p_i(k)$ [6]. Had they not succeeded receiving information from other clients, the peer tries to download it directly from the server, for which they stand a chance of $\tilde{p}(k)$ estimated previously at (4.7). Therefore, the probability of a peer in state i at time instant k to advance to the next state is:

$$\pi_i(k) = p_i(k) + (1 - p_i(k))\tilde{p}(k) \tag{4.9}$$

With this knowledge and the awareness of the peer birth rate $\mathscr{A}$ and the probability P_{stay} of seeders (peers in state M) staying in the system for another time interval, a recursive formula of the change of the state vector

[6] Calculation of $p_i(k)$ depends on the applied cache strategy and is discussed later on in other subsections.

$\underline{n}(k)$ can be deduced [7]:

$$\begin{cases} n_i(k+1) = \underbrace{\pi_{i-1}(k) \cdot n_{i-1}(k)}_{\substack{\text{peers advancing} \\ \text{from state } i-1 \text{ to } i}} + \underbrace{(1-\pi_i(k)) \cdot n_i(k)}_{\substack{\text{peers remaining} \\ \text{in state } i}} & \forall i = 2, \ldots, M-1 \\ n_1(k+1) = (1-\pi_1(k)) \cdot n_1(k) + \underbrace{\mathscr{A}}_{\text{new peers}} \\ n_M(k+1) = \pi_{M-1}(k) \cdot n_{M-1}(k) + \underbrace{P_{stay} \cdot n_M(k)}_{\substack{\text{seeders staying} \\ \text{for antoher interval}}} \end{cases} \tag{4.10}$$

The only thing that still needs to be calculated is the value $p_i(k)$, i.e. the probability of a peer in state i advancing to state $i+1$ by gaining the information from block $i+1$ purely from fellow peers. This number, however, depends on the applied cache strategy. We assume that all of the peers are equipped by the same storing mechanism, by the same cache strategy[8]. In the forthcoming subsections we investigate four major cache strategies, realising a formula of $p_i(k)$:

- deterministic K packet strategy, where the peer only stores the last K packets (note that K is the size of a single block);
- deterministic V block strategy, where the peer stores the packets from the last $V \in \{1, 2, \ldots\}$ blocks;
- unlimited cache strategy, where peers store all previously received data;
- random $V \cdot K$ packet strategy, where the peer stores a number of packets corresponding to V blocks and packets are selected randomly subject to uniform distribution among the already downloaded packets.

In the following subsections we denote the distribution function of the uniform distribution by Ψ:

$$\Psi(u) = \begin{cases} 0, & \text{if } u < 0 \\ u, & \text{if } u \in [0,1) \\ 1, & \text{if } u \geq 1 \end{cases}$$

[7] While constructing the formulas, we relied on the following basic idea: given m objects to sort out and each of the objects are independently chosen to be sorted out with a probability of p, then on average $m \cdot p$ objects are going to be chosen. In other words, the mean of binomial random variable with parameters m and p is $m \cdot p$.

[8] which should not be the case when size of the playback buffer or cache can be chosen by clients

4.1 Population model for deterministic K packet strategy

On one side, in case of deterministic K packet strategy peers will store the last K packets that were received, i.e. the information from the last block. Strictly speaking, a peer in state i at time instant k is going to cache the information from data block i as long as they advance to the next state. Should a peer in state i advance to the next state in time instant $k' > k$, they will no longer possess the information from block i at the new time instant. On the other side, peers in state i may only download information from block $i+1$ until they advance to state $i+1$ and in one time interval they can only download an amount of information corresponding to exactly one block. Therefore, download can only be made from peers in state $i+1$, since it is only them that store relevant information for peers in state i. Consequently, the following formula holds:

$$p_i(k) = P(\xi_{i,j}(k) = 0) = \Psi\left(\frac{n_{i+1}(k)}{n_i(k)}\right)$$

4.2 Population model for deterministic V block strategy

In this case, peers will store the information from the last V blocks, i.e. they will keep $V \cdot K$ packets. According to previous explanations, peers in state i are now able to download the desired information from peers found in states $i+1, i+2, \ldots, i+V$, if there exist that many peers. Consequently,

$$p_i(k) = P(\xi_{i,j}(k) = 0) = \Psi\left(\frac{\sum_{j=i+1}^{\max\{i+V,M\}} n_j(k)}{n_i(k)}\right)$$

4.3 Population model for unlimited cache strategy

In this case, peers store all previously received data. In that case, peers in state i may acquire the required information from any peers in a more advanced state. Consequently,

$$p_i(k) = P(\xi_{i,j}(k) = 0) = \Psi\left(\frac{\sum_{j=i+1}^{M} n_j(k)}{n_i(k)}\right)$$

4.4 Population model for random V block strategy

In this case, peers store exactly $V \cdot K$ packets from the previously received information and packets are selected randomly subject to uniform distribution among the already downloaded packets.

Firstly, we calculate the probability that a peer in state $j > i$ has exactly $S \in \{0, 1, \ldots, K\}$ relevant packets for the peer in state i (see figure 5). The $V \cdot K$ packets stored by the peer in state j are spread uniformly among blocks $2, 3, \ldots, j$, therefore every single combination has the same probability to emerge. There is no need to store information from 1, because, as we previously pointed out, newborn peers arrive immediately in state 1, they did not need to download actual content by that time.

Let us suppose that $j \geq V + 2$. Then peer in state j cannot cache every information from blocks $2, 3, \ldots, j$, because there are now $j - 1 \geq V + 1$ blocks. There are $\binom{(j-1)K}{VK}$ possibilities to choose the packets stored by the peer in state j. Useful combinations are those where exactly S packets are selected from block $i + 1$ (since this is the relevant block for peer in state i). The number of useful cases is: $\binom{K}{S} \cdot \binom{(j-2)K}{VK-S}$. Therefore, if we denote by $p^{(s)}_{i \leftarrow j}(k)$ the probability of the peer in state j having exactly S relevant packets for the peer in state i and by $\omega_{i \leftarrow j}(k)$ the number of relevant packets the peer in state j has for the peer in state i, then the following equalities hold:

$$p^{(s)}_{i \leftarrow j}(k) = \frac{\binom{K}{S} \cdot \binom{(j-2)K}{VK-S}}{\binom{(j-1)K}{VK}}$$

$$\mathbb{E}\left[\omega_{i \leftarrow j}(k)\right] = \sum_{S=0}^{K} S \cdot p^{(s)}_{i \leftarrow j}(k)$$

Therefore,

$$\mathbb{E}\left[\omega_{i \leftarrow j}(k)\right] = \sum_{S=0}^{K} S \cdot \frac{\binom{K}{S} \cdot \binom{(j-2)K}{VK-S}}{\binom{(j-1)K}{VK}} \tag{4.11}$$

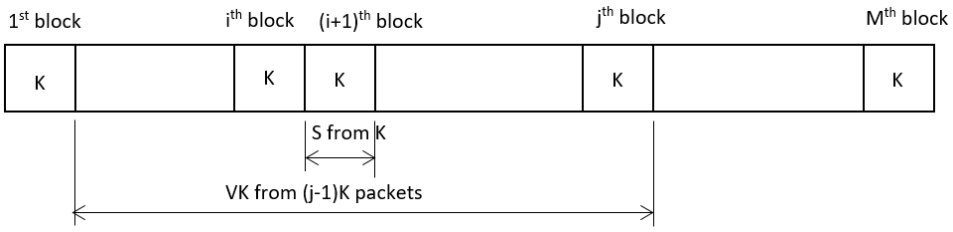

Figure 5: The bar representing the blocks of the file.

We may simplify (4.11) this way:

$$\begin{aligned}\mathbb{E}\left[\omega_{i\leftarrow j}(k)\right] &= \frac{1}{\binom{(j-1)K}{VK}} \cdot \sum_{S=0}^{K} S \cdot \binom{K}{S} \cdot \binom{(j-2)K}{VK-S} \\ &= \frac{K}{\binom{(j-1)K}{VK}} \cdot \sum_{S=1}^{K} \binom{K-1}{S-1} \cdot \binom{(j-2)K}{VK-S}\end{aligned}$$

The sum is equivalent to the following problem: in how many ways we may choose from two piles of objects, one with $K-1$ elements, the other with $(j-2)K$ elements, in such a way that we select exactly $(S-1)+(VK-S)=(VK-1)$ elements. The very same result is obtained when we determine the number of ways, we may select $VK-1$ elements from $(K-1)+(j-2)K=(j-1)K-1$ objects. Consequently,

$$\begin{aligned}\mathbb{E}\left[\omega_{i\leftarrow j}(k)\right] &= \frac{K}{\binom{(j-1)K}{VK}} \cdot \binom{(j-1)K-1}{VK-1} \\ &= \frac{K}{\frac{(j-1)K}{VK} \cdot \binom{(j-1)K-1}{VK-1}} \cdot \binom{(j-1)K-1}{VK-1} \\ &= \frac{V}{j-1} \cdot K, \qquad \forall j \geq V+2\end{aligned}$$

For $j \leq V+1$ peer in state j should keep every single packet till block j. Therefore, he may serve peer in state $i+1$ with all the K packets from block $i+1$. That means, $\mathbb{E}\left[\omega_{i\leftarrow j}(k)\right]=K \qquad \forall j \leq V+1$.

Let $\Omega_i(k)$ denote the number of blocks that peers in states $j+1,\ldots,M$ have for a peer in state i. Then, we find that:

$$\mathbb{E}\left[\Omega_i(k)\right] = \sum_{j=i+1}^{M} n_j(k) \cdot \frac{\mathbb{E}\left[\omega_{i\leftarrow j}(k)\right]}{K}$$

Consequently, the probability of a peer in state i to advance to state $i+1$ is:

$$p_i(k) = \Psi\left(\frac{1}{n_i(k)}\mathbb{E}\left[\Omega_i(k)\right]\right)$$

Due to the complexity of the equations, we resort to numerical methods in the next section in order to collect statistics illustrating the performance of the system.

5 Numerical results and analysis

In this section we provide an extensive numerical analysis and performance evaluation of the download process. The simulation package is capable of analysing the performance in the case of different peer birth rates and cache strategies. The software calculates the distribution of clients in the system for the entire time of interest. Besides that, we could plot and analyse the entropy of peer distribution in different download states, which provides a major quality characteristic of the system. Our model is once again outlined here as was detailed in section 4 at 4.1.

$$S =< \mathcal{F}(M,K), C, cache\ strategy, \mathscr{A}, P_{stay} >$$

5.1 Comparison of cache strategies

Tables in figure 6 depict the results of our simulation. In all figures we applied the following choice of parameters: the number of blocks was chosen to be $M = 10$, the number of packets within a block $K = 20$. Seeders (peers in state M) remain in the system for another time instant with a probability of $P_{stay} = 0.3$. Peer birth rate $\mathscr{A}$ was estimated by a Poisson random variable whose λ parameter was changed during the course of benchmark [9]. Different tables were constructed for different cache strategies.

Our main concern, when experimenting with our model, was to estimate the required threshold of the server capacity C at the previously given parameters and at different values of peer birth rate applying different cache strategies. The tables were constructed according to the followings:

- each of the different cache strategies is associated with different tables;
- within a table the first column represents the λ parameter of the peer birth rate (which also represents the mean of the Poisson distribution), varying between values $50, 100, 300$;
- the second column illustrates the server capacity. We attempted to determine the threshold under which the system would collapse with reasonable chance and above which the system stays robust in all of the cases;

[9] A Poisson random variable is a discrete random variable with values falling in $\{0, 1, 2, \ldots\}$ and with a distribution of the following form: $P(k) = \frac{\lambda_k}{k!} \cdot e^{-\lambda}$. It is crucial to note that the mean of such a distribution is λ. This means that when we are referring to Poisson nature peer birth rate with parameter λ, then we are actually talking about a birth rate with a mean of λ.

Deterministic K Packet Strategy				
Lambda	C	success	out of	%
50	80	6	20	0.3
50	85	8	20	0.4
50	90	13	20	0.65
50	95	16	20	0.8
50	100	19	20	0.95
100	160	12	20	0.6
100	170	15	20	0.75
100	180	16	20	0.8
100	190	18	20	0.9
100	200	20	20	1
300	900	17	20	0.85
300	950	18	20	0.9
300	1000	19	20	0.95
300	1050	19	20	0.95
300	2000	19	20	0.95

(a)

RandomVBlockStrategy V=3				
Lambda	C	success	out of	%
50	80	0	10	0
50	100	7	10	0.7
50	105	10	10	1
50	110	9	10	0.9
50	120	10	10	1
100	180	3	10	0.3
100	190	4	10	0.4
100	200	9	10	0.9
100	210	10	10	1
100	220	10	10	1
300	1200	0	10	0
300	1300	0	10	0
300	1400	6	10	0.6
300	1500	9	10	0.9
300	1800	10	10	1

(b)

DeterministicVBlockStrategy V=3				
Lambda	C	success	out of	%
50	60	14	20	0.7
50	65	17	20	0.85
50	70	18	20	0.9
50	75	20	20	1
50	80	20	20	1
100	110	10	20	0.5
100	120	17	20	0.85
100	130	18	20	0.9
100	140	20	20	1
100	150	20	20	1
300	750	5	20	0.25
300	775	8	20	0.4
300	800	18	20	0.9
300	850	19	20	0.95
300	900	20	20	1

(c)

UnlimitedCacheStrategy				
Lambda	C	success	out of	%
50	55	7	20	0.35
50	60	12	20	0.6
50	65	15	20	0.75
50	70	19	20	0.95
50	75	20	20	1
100	110	10	20	0.5
100	120	15	20	0.75
100	130	19	20	0.95
100	140	20	20	1
100	150	20	20	1
300	750	6	20	0.3
300	775	10	20	0.5
300	800	18	20	0.9
300	850	20	20	1
300	900	20	20	1

(d)

Figure 6: Results of the simulation for the following parameters: $M = 10$, $K = 20$, $P_{stay} = 0.3$, the peer birth rate follows a Poisson distribution.

- the following three columns represent the outcome of the benchmark. Column "out of" shows the number of tests conducted with the given parameters and column "success" conveys the result, i.e. how many tests are treated as successful. Soon the definition of a successful test is described. Column "%" represents the proportion of the previous columns.

- In the case of the tables 6a, 6b, 6c we calculated the state vector $\underline{n}(k)$ of the system for the first $I = 500$ time instants, in case of the 6d we have done similarly until $I = 200$ because of the complexity of the test.

- We considered a test successful provided the entropy of the state vector at the last time instant calculated was greater than 0.85. That means that a test was successful if and only if

$$\mathcal{H}(\underline{n}(k)) = \sum_{i=1}^{M} \frac{n_i(k)}{\sum_{j=1}^{M} n_j(k)} \cdot \ln\left(\frac{\sum_{j=1}^{M} n_j(k)}{n_i(k)}\right) > 0.85 \tag{5.1}$$

  where k denotes the last time instant the simulation reached.

It should be outlined, that $K = 20, M = 10$ is a fair choice. BitTorrent-like protocols use a packet size of $32kB$. In that case, the amount of data in a block is $2.56Mb$ and the amount of data available in advance is $51.2Mb$. It is assumed that FullHD quality IPTV streams consume around a $10Mbs$ network bandwidth. In that case, we could watch our live stream with an approximately 5 second latency [10].

As far as the results are concerned, we may conclude that cache strategies using larger cache size will largely outperform strategies making use of less cache, but after a certain size the performance cannot be further increased. Comparing 6a and 6c, one can easily observe that while deterministic K packet strategy easily consumes a 2000-capacity server with a 300-mean peer birth, the unlimited cache strategy will reduce this bound to 900. This means a more than 50% reduction in the server load. In case of 100 peers the necessary capacities are 200 versus 150. Therefore, unlimited cache strategy performs reasonably well for low peer birth rates, as well.

However, there is astonishing similarity between the unlimited cache strategy and the deterministic V block strategy with $V = 3$ blocks. The two cases produce almost the same result The unlimited cache strategy can seldom outperform his counterpart. This provides a useful result, since peers will not need to store the received file in its entirety, but they only need to allocate a previously known amount of space for caching. Note that real

[10] Although live streaming and IPTV watching do not necessarily mean the same burden for the network, this is still the right scale of parameters to discuss.

unlimited cache strategy cannot be implemented in any way, but, according to our findings, it will not need to be.

One can also analyse and compare the results of the deterministic K packet strategy and deterministic V block strategy. It must be remarked that the random strategy cannot outperform the deterministic strategies. Even though consuming a cache size corresponding to three blocks, it produces almost the same result as the deterministic K packet strategy, staying well behind the other two storing mechanisms. The problem with the design is that peers in lower states are served excessively well by advanced peers, who, in return, however, may be poorly served by seeders who have kept packets for peers from all age brackets.

5.2 Relationship between peer birth rate and minimum server capacity needed

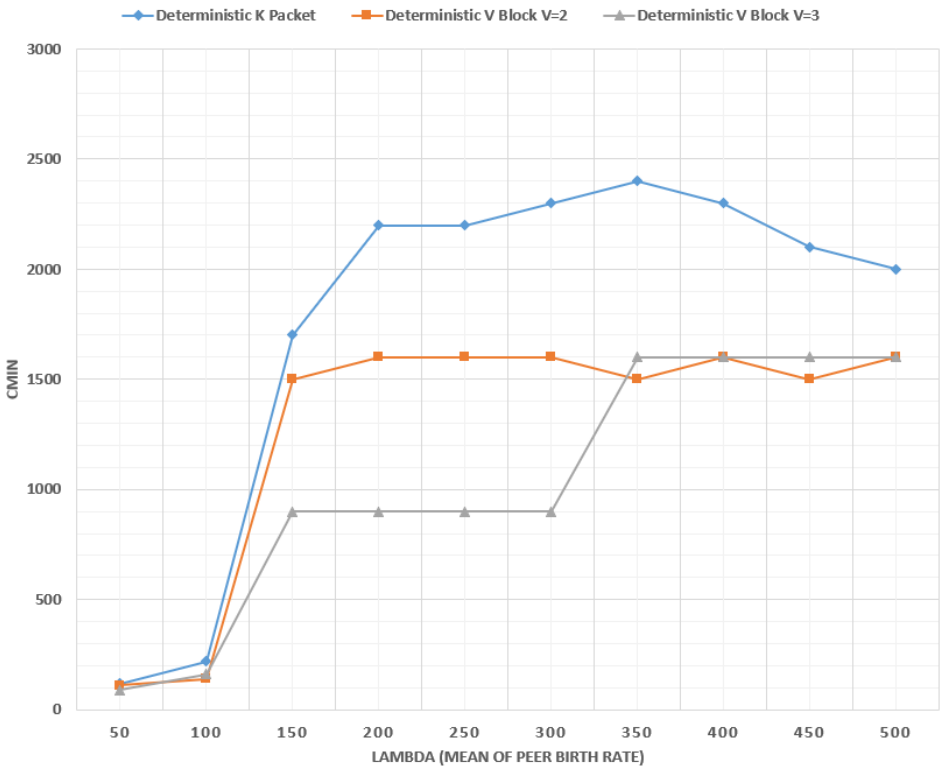

Figure 7: Relationship between the λ parameter of the peer birth rate and the C_{min} server capacity in case of three different cache strategies. Parameters: $K = 20$, $M = 10$, $P_{stay} = 0.3$, number of time instants considered: $I = 500$, number of tests $T = 20$

In this subsection we aim to reveal the relationship between the mean

of the peer birth rate and the minimum server capacity needed, so that the server would not collapse being heavily overloaded. Basic parameters were kept from the previous investigations. The major difference is that we plotted the minimum server capacities in case of three different cache strategies (deterministic K packet scenario, deterministic V block scenario with V=2 and V=3) on the same chart allowing a detailed comparison. The minimum server capacity represents the estimation of the lowest capacity at which file sharing scenarios will almost terminate with a fairly large entropy. In our case we determined C_{min} in such a way that it would be the lowest server capacity where every single test produced a last state of vector with the entropy greater than 0.85 (see formula at (5.1)).

First and foremost, the advantage of the larger cache strategy becomes clearly visible in the interval $\lambda \in [100, 300]$, where the minimum server capacity decreases almost proportionately to the cache size. However, it is interesting to see that such a relationship is disturbed at larger means: at $\lambda = 350$ the deterministic V block strategies with V=2 and V=3 are almost of the same quality, and after that none of them can reasonably perform better than the other one.

Secondly, it is surprising how large the deviation in required server capacity is, when $\lambda \in [100, 150]$. With a handful of addition to the mean, the required server capacity increases beyond one thousand.

6 Conclusion

In this work we studied the performance of P2P-like systems using a general stochastic model. In the first part we have analysed some already existing models of P2P protocols. Then our emphasis was focused on how the different parameter settings affect the performance. The key contribution of this work is the introduction of a stochastic model, which can capture the dynamic behaviour of the system. This dynamics describes the peer progress and is characterised by a state vector the components of which indicate the peer population in different download stages. After normalisation, the entropy defined over this vector will measure how "smooth" the download is or if there is a bottleneck stage, where peers are waiting for new packets without being served by other peers. This dynamics can shed light on how the different caching strategies and server capacities will affect the system performance. A detailed numerical analysis was conducted which gave rise to the following conclusions: (i) the deterministic K packet caching strategy (i.e. keeping the last K packet) perform better than random ones (keeping K packets which are randomly selected subject to uniform distribution), (ii) if the cache memory size is large, then the performance of the download process does not depend on the peer population any longer.

The obtained results can increase the quality of video streaming applications. Future work in the field will involve establishing the relationship between the individual peer progress and the peer age. Besides that, it also remains to be seen how network coding may further enhance the performance.
Finally, let me underline the help of Mr Patirk Braun, PhD student at the Department of Automation and Applied Informatics whose work has pioneered this contribution.

References

[1] S. Tewari and L. Kleinrock: *Analytical Model for BitTorrent-based Live Video Streaming*, in Consumer Communications and Networking Conference, 2007. CCNC 2007. 4th IEEE

[2] Y. Yue, C. Lin and Z. Tan: *Analyzing the Performance and Fairness of BitTorrent-like Networks Using a General Fluid Model*, in Global Telecommunications Conference, 2006. GLOBECOM '06. IEEE

[3] X. Hei, C. Liang, J. Liang, Y. Liu and K.W. Ross *Insights into PPLive: A measurement study of a large-scale P2P IPTV system*, in Workshop on Internet Protocol TV (IPTV) services over World Wide Web in conjunction with WWW2006, May 2006

[4] *The Zettabyte Era: Trends and Analysis*, in Cisco VNI Global IP Traffic Forecast, June 2017

[5] *PPLive*, http://www.pplive.com/en/about.html

[6] *BitTorrent*, http://www.bittorent.org/protocol.html

[7] B. Cohen, : *Incentives Build Robustness in BitTorrent*, in IPTPS, February 2003

[8] Z. Ma, D. Qiu: *A Novel Optimistic Unchoking Algorithm for BitTorrent*, in Consumer Communications and Networking Conference, 2009. CCNC 2009. 6th IEEE, 10-13 Jan 2009

[9] D. Qiu and R. Srikant: *Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks*, in Proceedings of ACM SIGCOMM, Aug. 2004

[10] C. Barakat and I. Pratt *Passive and Active Network Measurement*, in 5th International Workshop, PAM 2004, Antibes Juan-les-Pins, France, April 19-20, 2004

A Appendix: Simulation Package

In order to numerically analyse the stochastic model, we developed a complex simulation package on .NET C# basis. Figure 8 outlines the package representing the model of the architecture. This architecture includes mostly the logical view of the software and it does not get involved with showing data on a user interface (that is the role of the *View* package). The core class of the architecture is the *AbstractP2PDownloadStateModel*, which is responsible for maintaining the statevector $\underline{n}(k)$ during the calculations. The software is still open for other population models, therefore this class is abstract and only its descendent, *BirthDeathRateP2PDownloadStateModel* implements the usage of the peer birth rate $\mathscr{A}$ or the seeder staying probability P_{stay}. Here the use of the template method pattern offers us a flexible solution. The *CacheStrategy* interface represents an incarnation of the strategy pattern invoked here, to provide interchangeability of different cache strategies. The *AbstractP2PDownloadStateModel* is also responsible for maintaining the logarithmic generator function and to calculate the desired minimum point outlined previously at 1. For implementing this we introduced a singleton *Optimisation* class that would support experimenting with different optimisation algorithm without affecting directly the model logic. Consequently, our model follows some basic object oriented principles for the pure purpose of providing future extensions, maintainability.

What is more, a view package has been produced to visualise numerically determined results. Our current version of the software supports setting parameters and analyse the outcome of single session, which is depicted in figure 9. The left hand side chart illustrates the state vector $\underline{n}(k) = (n_1(k), n_2(k), \ldots, n_M(k))$ at a time instant we set with trackbar in the middle (gray) section. The right hand side figure plots the change on entropy during the course of experiment: $\mathcal{H}(\underline{n}(k)) = \sum_{i=1}^{M} \frac{n_i(k)}{\sum_{j=1}^{M} n_j(k)} \cdot \ln\left(\frac{\sum_{j=1}^{M} n_j(k)}{n_i(k)}\right)$. The red bar crossing horizontally the chart represent the maximum value the entropy can hold, namely $\mathcal{H}_{max} = \ln(M)$.

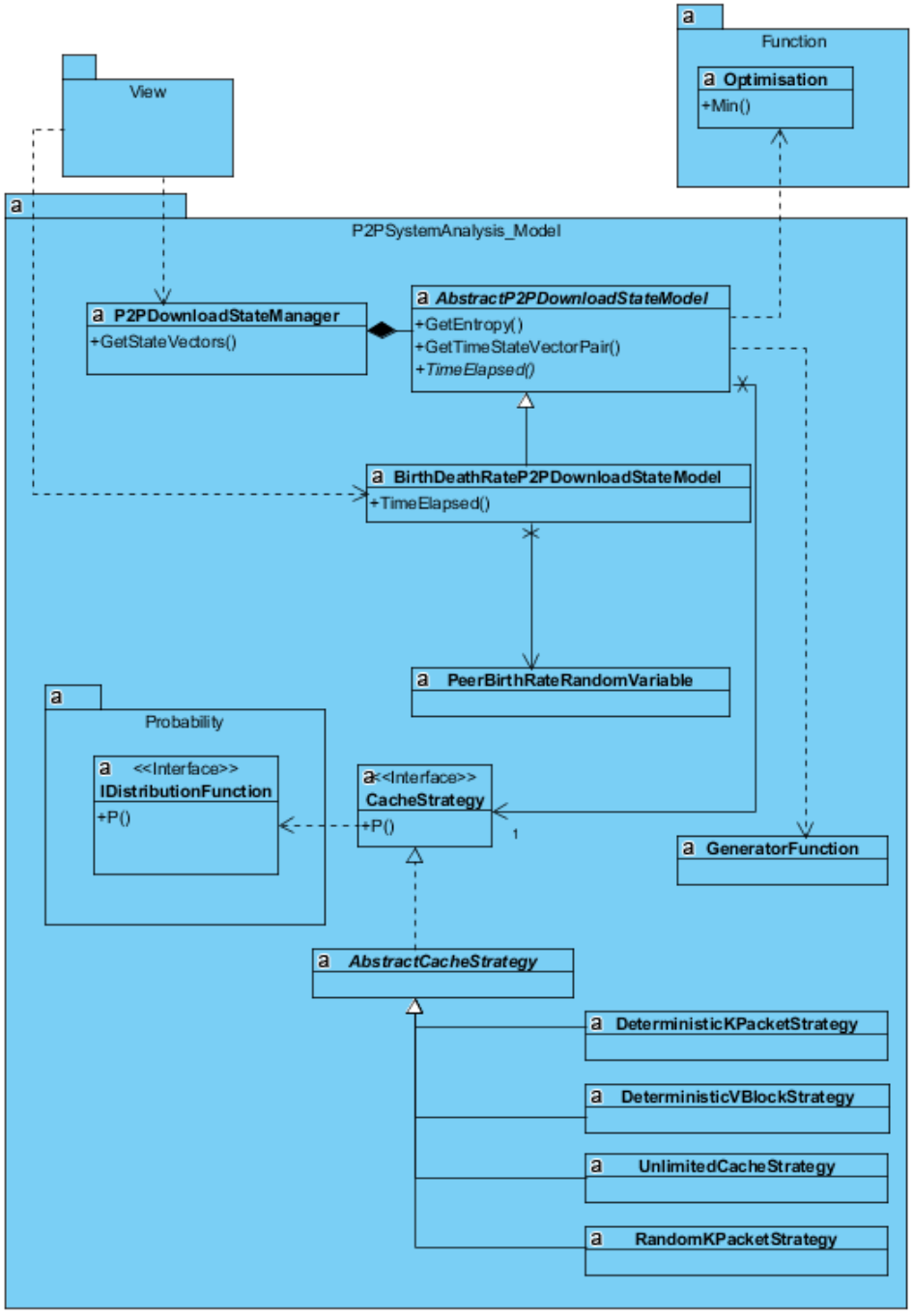

Figure 8: Software architecture of the model of the simulation package

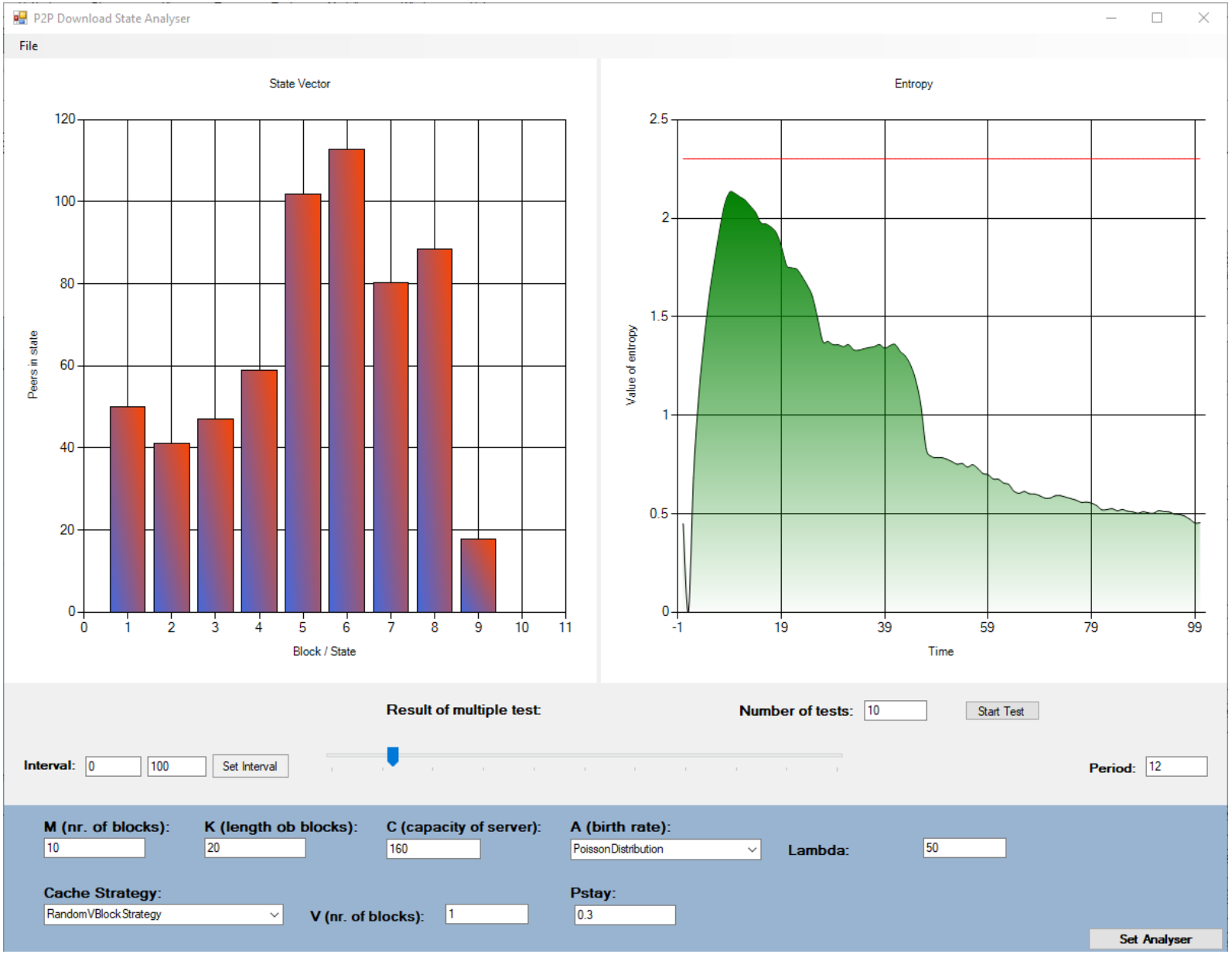

Figure 9: User interface of the simulation package