Knapsack Problem-based Piece-Picking Algorithms for Layered Content in Peer-to-Peer Networks

Michael Eberhard[1], Tibor Szkaliczki[2], Hermann Hellwagner[1], László Szobonya[2], and Christian Timmerer[1]

[1]ITEC, MMC – Klagenfurt University, Austria, *firstname.lastname@itec.uni-klu.ac.at*

[2]eLearning Department – Computer and Automation Research Institute of the Hungarian Academy of Sciences, Hungary, *sztibor@sztaki.hu*, *szobonya@sztaki.hu*

ABSTRACT

The distribution of layered content over peer-to-peer networks becomes more important today as the users are consuming the content on terminals with various display capabilities and different network connections. For single-layer content distribution, the piece-picking algorithm only needs to ensure that content pieces are downloaded in time for display. When layered content is distributed over a peer-to-peer network, the piece-picking algorithm needs to be modified to ensure that the best possible quality is displayed while all desired pieces still have to be received before their deadline expires. In this paper, the piece-picking problem for layered content is analyzed and a number of piece-picking algorithms for layered content based on the solutions for the knapsack problem are presented. Furthermore, an evaluation of these algorithms is performed and possible applications are discussed.

Categories and Subject Descriptors

I.1.2 [**Computing Methodologies**]: Algorithms – *Analysis of Algorithms*.

General Terms

Algorithms, Measurement, Performance, Design.

Keywords

Knapsack Problem, Layered/Scalable Content, Piece-Picking, Piece Utility Calculation.

1. INTRODUCTION

As the popularity of streaming multimedia data over peer-to-peer (P2P) networks is constantly increasing nowadays, the users have access to network connections with varying bandwidth capabilities and are consuming the content on diverse terminals. Thus, it is important to provide the content in a number of different qualities to ensure that the user can consume the content in a suitable quality. The traditional approach for this problem is to provide the same content in different qualities encoded in different files. Although this approach works fine for most cases, it makes the sharing process less efficient, as only users consuming exactly the same quality can share data with each other. Solutions for layered video coding, where the different qualities are provided within a single bitstream, are better suited, as all peers interested in this content can exchange the base layer, and the optional enhancement layers can be shared with all peers interested in the same or higher quality. Additionally, the support of layered codecs can significantly reduce the start-up delay when streaming content over P2P.

When content is distributed over P2P networks, the piece-picking algorithm ensures that the desired pieces are downloaded before their deadline expires. This is especially important for live streaming or Video on Demand (VoD) scenarios, where it is essential that the pieces are received in time for display in the video player. For the distribution of layered content, the piece-picking algorithm needs to be modified to consider, in addition to the deadline, the layer of the piece. The main goal of the layered piece-picking algorithm is to ensure that all pieces are received in time for playback while trying to provide the best possible quality for the available bandwidth at every time instance. Additionally, frequent quality switches should be avoided as such switches are usually more disturbing for the user than watching the video at slightly lower, but constant quality. The problem of finding the best trade-off between smooth playback and displaying the best possible quality, while also trying to avoid quality switches, represents a very challenging optimization problem. In this paper, a number of different algorithms for the piece-picking of layered content that address this optimization problem are described and evaluated. Although the algorithms are codec-agnostic, the Scalable Video Coding (SVC) extension of the Advanced Video Coding (AVC) standard [1] has been utilized for our implementation work.

To find a feasible piece-picking algorithm for layered content, several approaches have been investigated in [2]. The piece-picking problem is very close to the knapsack problem (*KP*) [3], which is a well-known problem in combinatorial optimization. Therefore, the algorithms for the *KP* including solutions utilizing dynamic programming and greedy methods can be adapted to solve the piece-picking problem. In the evaluation section, the knapsack-related algorithms are evaluated and compared to a baseline algorithm using the provided simulation framework.

The remainder of this paper is organized as follows. In Section 2 the related work is discussed. Section 3 provides an introduction to the layered piece-picking problem. In Section 4 a detailed description of the piece selection algorithms addressed in this paper is provided. Finally, in Section 5 the presented piece-picking algorithms are evaluated and compared to each other, while Section 6 concludes the paper.

2. RELATED WORK

The distribution of scalable content in P2P systems has been a popular research topic in recent years. There are already a number of P2P systems with SVC support and some of them, like LayerP2P [4], propose very well defined solutions for the distribution of scalable content in P2P systems. However, these P2P prototype systems are implemented from scratch with the intention to support scalability, and do not offer compatibility to

AVSTP2P'10, October 29, 2010, Firenze, Italy.

	t-1	t	t+1	t+2	t+3	t+4	t+5	t+6	t+7	t+8	t+9
EL3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
EL2	1.0	1.0	1.0	1.0	0.7	0.3	0.0	0.0	0.0	0.0	0.0
EL1	1.0	1.0	1.0	1.0	1.0	0.8	0.5	0.0	0.0	0.0	0.0
BL	1.0	1.0	1.0	1.0	1.0	1.0	0.9	0.5	0.0	0.0	0.0

Figure 1. Sliding Window.

already existing P2P systems which provide a large user base and huge amounts of content. The focus of the work presented in this paper is to integrate the support of layered content into an already existing P2P system, the *NextShare* system [5], which is backwards compatible to the *Bittorrent* protocol. Thus, the architectural choices were made and the algorithms were developed taking the requirements of the *Bittorrent* protocol into account while ensuring that the algorithms can be easily integrated into a codec-agnostic *Bittorrent*-based P2P system.

3. LAYERED PIECE-PICKING PROBLEM

In this section the layered piece-picking problem with regard to the *NextShare* P2P system [5] is described. The *NextShare* P2P system is a fully decentralized P2P system which is based on a modified version of the *Bittorrent* protocol (supporting live streaming and VoD). The system is agnostic of the transmitted content and is intended for sharing all kinds of data and audiovisual content. Thus, when investigating different piece-picking algorithms, it had to be taken into account that the algorithms need to be integrated into an existing *Bittorrent*-based P2P system, which imposes a number of requirements. When content is distributed in *Bittorrent*, it is split into pieces of fixed size. To perform a mapping of the layered content to pieces of fixed size, the audiovisual content is firstly provided at a constant bitrate. The content is then split into pieces which contain a fixed number of frames for each layer. Depending on the bitrate of the layers, the size of the pieces for each layer can vary. Thus, the pieces for each layer can be mapped to a different number of actual *Bittorrent* pieces. During the piece-picking process the mapping is still considered, i.e., if one piece is selected for download all corresponding *Bittorrent* pieces need to be selected for download. Additionally, only the layer-based scalability of SVC is considered initially, as a client can only decide on piece-level which quality to download. A detailed description of the architectural choices for the integration of layered content in the *NextShare* architecture and more information on the layer and piece structure within *NextShare* are provided in [5].

As mentioned before, the piece-picking algorithm for single-layer content and live streaming or VoD scenarios assigns the priority only based on the deadline of the pieces. For layered content, the layer of the pieces also needs to be considered. While the priority settings based on the deadline remain the same as for single-layer content, the layer-based priority settings assign a higher priority to lower layers. The reason for this assignment is that higher layer pieces depend on lower layers for decoding. In addition to the deadline and the layer, also the available pieces from the previous timeslots need to be considered to avoid frequent quality switches.

Although the calculation of the piece utilities is rather complex, such a calculation only needs to be performed for pieces with a deadline in the near future (the pieces within the sliding window, described below). For the pieces with a later deadline, the rarest-first strategy as employed in the original *Bittorrent* protocol is usually the best choice. An illustration of such a sliding window is provided in Figure 1.

In Figure 1 the rows illustrate the layers and the columns represent the timeslots. It should be noted that the size of the timeslots is constant, as each piece contains a fixed number of frames. The range of the sliding window is illustrated by the rectangular border. The numbers in the cells represent the download status, i.e., *1.0* indicates that the piece has been successfully downloaded and *0.0* indicates that the download has not started yet. As mentioned before, the cells in the sliding window represent pieces, which can be mapped to a different number of actual *Bittorrent* pieces depending on the bitrate of each layer. In the situation illustrated in Figure 1 the piece-picking algorithm has to decide which pieces need to be downloaded for the upcoming timeslots *t+1* to *t+8*. When taking this decision, the algorithm has to find the best trade-off between smooth playback (no freezing or quality switching) and trying to provide the best possible quality to the user. Thus, the algorithm could decide to download the base layer pieces for *t+7* and *t+8* first, to avoid player freezing even if the network conditions get worse in the future. Another possibility would be to focus on downloading the second enhancement layer for *t+5*, and subsequently the first and second enhancement layer for *t+6*, to ensure that the pieces required for the current playback quality arrive in time. If the network conditions have improved over the last time instances, the algorithm might even decide to increase the current playback quality, e.g., by downloading the third enhancement layer for *t+1* and *t+2*. When actually applied, the algorithm will have to consider all of these possibilities and find the best possible solution for the current network conditions and user preferences.

The initial filling of the sliding window is performed during the start-up phase. The start-up phase consists of a specific number of timeslots (usually around half or all of the timeslots of the sliding window) during which the initial download window is filled. The start-up algorithm starts to download the pieces of the lowest layer for all of its timeslots and then continues with the download of the pieces for the next higher layers, as long as there is time remaining in the start-up phase. After the start-up phase is finished, the highest layer for which all of the pieces within the initial download window have been downloaded is taken as the initial target download quality.

In general, all pieces within the sliding window that are not currently being downloaded are considered for the piece-picking process. However, if pieces are unlikely to be received in time, their download can be stopped even before the deadline expires. Additionally, some pieces might be downloaded from more than one neighbour peer if they have a high priority and the deadline is already close. These special cases are considered when preparing the piece queue for the piece-picking algorithms. The algorithms subsequently select the most useful pieces from this queue.

It should be noted that additionally to the piece selection process, all pieces selected for download have to be assigned to a suitable neighbour peer for download. This peer selection process is usually performed after the piece selection and assigns the download capacity of the neighbour peers to the pieces according to their utility (i.e., the piece with the highest utility is downloaded from the neighbour peer with the best download capability).

Table 1. Notations.

t_i	the ith timeslot of the stream
t_k	the kth decision point during the download of the stream
l_j	the jth layer of the stream
n_l	the lth neighbour node (neighbour peer)
m	the number of timeslots within the sliding window
n	the number of layers within the sliding window
z	the number of neighbour peers
$p_{i\,j}$	a piece at timeslot t_i and layer l_j
d_j	the distortion reduction importance of a piece at layer l_j
$pr_{i\,j\,k\,l}$	the probability at decision point t_k that the piece $p_{i\,j}$ will be downloaded until its timeslot t_j from neighbour node n_l
$wp_{i\,j\,k\,l}$	the weighted probability at decision point t_k that the piece $p_{i\,j}$ and all the pieces it depends on will be downloaded until its timeslot t_j from neighbour node n_l
$wp_{i\,j\,k}$	the weighted probability at decision point t_k that one of the downloads of piece $p_{i\,j}$ and all the pieces it depends on will be successful until its timeslot t_j
$u_{i\,j\,k}$	the utility of the piece $p_{i\,j}$ at decision point t_k
α	the urgency weighting, used to influence the ratio between urgency and distortion reduction of a piece
c_j	the required bandwidth for transmission of a piece at layer l_j
S	the maximum available download bandwidth
$wu_{i\,j\,k}$	the weighted utility of the piece $p_{i\,j}$ at decision point t_k
$x_{i\,j\,k}$	indicates whether piece $p_{i\,j}$ is selected for download at decision point t_k (*1* if selected, *0* otherwise)

4. LAYERED PIECE-PICKING ALGORITHMS

In this section different algorithms for solving the piece-picking problem are investigated. Before the actual discussion of the algorithms, the utility calculation, which is used by all algorithms, and the *KP*, which provides the basis for the algorithms, are described. An overview of the notations utilized in the following sections of this paper is presented in Table 1.

4.1 Utility Calculation

The calculation of the utility value is used by the piece selection algorithms in order to determine which pieces to select for download (pieces with higher utility are downloaded with higher priority). The utility of a piece is based on its layer, deadline, and download probability. Thus, to calculate the utility of a piece, it is firstly necessary to calculate the weighted download probability that the piece is received in time and is useful (a piece is only useful if the pieces of all lower layers at the same timeslot are also available). The weighted download probability is defined as follows:

$$wp_{i\,j\,k\,l} = \prod_{j' \le j} \left(pr_{i\,j'\,k\,l}\right) \times pr_{i-1\,j\,k\,l} \tag{1}$$

To calculate the weighted download probability, the download probability for the actual piece is multiplied with the download probability for the pieces of all lower layers at the same timeslot. The download propability $pr_{i\,j\,k\,l}$ is calculated based on the remaining download size and the estimated download bandwidth from the neighbour peer (the estimated download bandwidth is available in the *NextShare* P2P system). Additionally, the result is multiplied with the download probability for the piece at the same layer at the previous timeslot, which prohibits the algorithm to implicitly switch to a higher quality. To avoid frequent quality switches, switches to a higher quality are not performed by the piece selection algorithm but are performed explicitly if the bandwidth is higher than expected over a number of timeslots. For this purpose a monitoring algorithm is used which utilizes the excess bandwidth to fill the buffer for the higher layers. If the bandwidth conditions remain constantly improved for some time, the algorithm switches to the higher layer and provides the initial buffer filling for the new layer(s).

The overall weighted download probability that summarizes the download probability from all neighbour peers (if the piece itself or pieces it depends on are downloaded from multiple neighbours) is subsequently defined as

$$wp_{i\,j\,k} = 1 - \prod_{l' \le z} \left(1 - wp_{i\,j\,k\,l'}\right) \tag{2}$$

where z specifies the number of neighbour nodes and the product in the formula specifies the probability that the piece is not received in time from any of the neighbour peers.

Based on (2) the utility is defined as follows:

$$u_{i\,j\,k} = \frac{d_j \times wp_{i\,j\,k}}{(t_j - t_k)^{\alpha}} \tag{3}$$

The general importance of a piece is defined as d_j, which describes its importance with regard to the distortion reduction. The importance is defined based on the distortion reduction, as each received piece reduces the distortion. Thus, the importance for the base layer pieces is the highest, as these pieces provide the biggest distortion reduction (from no content at all to the basic quality). The distortion reduction importance is multiplied with the weighted download probability and divided by its urgency (the number of timeslots remaining to finish the download). Finally, the parameter α is utilized to influence the ratio between the urgency of the piece and its distortion reduction importance.

To sort the pieces according to their utility, the cost for the transmission of a piece (the number of actual *Bittorrent* pieces it consists of) also needs to be taken into account. Thus, the weighted utility is defined as

$$wu_{i\,j\,k} = \frac{u_{i\,j\,k}}{c_j} \tag{4}$$

and is used to evaluate the utility of a piece based on the required bandwidth. As the piece size is constant for each layer, the cost values are only associated with the layers, but not with the timeslots.

4.2 The Knapsack Problem

In this section we firstly provide a formal definition of the piece selection problem and then relate it to the *KP*. The piece selection at decision point t_k can be defined formally as an optimization problem as follows:

Maximize

$$\sum u_{i\,j\,k} \times x_{i\,j\,k} \tag{5}$$

Subject to

$$\sum c_j \times x_{i\,j\,k} \le S \tag{6}$$

$$x_{i\,j\,k} \in \{0,1\} \tag{7}$$

$$x_{i\,j\,k} \le x_{i\,j-1\,k} \tag{8}$$

$$x_{i\,j\,k} \le x_{i-1\,j\,k} \tag{9}$$

The aim of the piece selection process is to maximize the total utility of the selected pieces (5). Constraint (6) expresses the limit on the total cost of the pieces, i.e., the required bandwidth for the selected pieces has to be lower than the available download bandwidth. Each piece can be either selected or not selected for download (7). Due to the dependency between the layers, a piece can be selected only if the piece in the lower layer is also selected (8). In order to avoid frequent quality switches, constraint (9) is introduced to ensure that a piece can be selected only if the piece in the preceding timeslot of the same layer is also selected for download. The switching between layers is performed by the monitoring algorithm, which only switches layers if the network conditions remain changed over a longer period. However, switches to lower layers are still implicitly possible, if there is not sufficient bandwidth available to select higher layer pieces.

4.3 Dynamic Programming for the Knapsack Problem

Exact solutions for the *KP* using dynamic programming have already been studied extensively in the literature [6]. This solution of the *KP*, from now on referred to as *DP*, does not consider the precedence constraints among the pieces, i.e., that the piece is useful only if the pieces it depends on are also selected. Its running time is $O(S \cdot m \cdot n)$.

When the *DP* algorithm is applied for piece-selection, it needs to be ensured that only useful pieces are selected. This can be either achieved when certain conditions regarding the utility and layer structure are fulfilled or by considering the dependencies explicitly in the algorithm. The conditions for the *DP* algorithm to select only useful pieces are provided below.

$$\left(t_{i'} \leq t_i \; and \; l_{j'} \leq l_j\right) \Longrightarrow \left(u_{i'j'k} \geq u_{ijk} \; and \; c_{j'} \leq c_j\right) \qquad (10)$$

(10) specifies that if a piece p_{ij} depends on piece $p_{ij'}$, its utility cannot be larger than the utility of the piece it depends on. This condition is in conformance with the definition of the distortion reduction importance (d_j), which specifies that pieces of lower layers have a higher importance. Similarly, the condition specifies that a piece $p_{i'j}$, which has an earlier deadline than a piece p_{ij}, always needs to have a higher utility. Again, this is in conformance with the definition of the utility (3), where the importance value is divided by the remaining timeslots (and fewer remaining timeslots result in a higher utility).

However, regarding the layer costs, condition (10) is very restrictive. To ensure that the optimal solution is found by the *DP* algorithm, the costs for lower layers always have to be lower or equal to the costs of the higher layers. Although this layer structure condition is often fulfilled and the *DP* algorithm still only selects useless pieces if no other pieces can be selected with the remaining bandwidth, it limits the applicability of the algorithm.

The *DP* algorithm can also be extended to select only useful pieces by considering the dependencies between the pieces explicitly. This can be done but its complexity then increases to $O(S \cdot m \cdot n^2)$. More details on the extension of the *DP* algorithm and its application to the piece-picking problem can be found in [2].

4.4 The Multiple-Choice Multi-Dimension Knapsack Problem

The dependency between the pieces does not have to be explicitly addressed if we consider the multiple-choice knapsack problem (*MCKP*). In this problem, there are several groups of items, each group representing one timeslot, and it is enough to choose only one item from each group. One item represents the piece sequence from the lowest layer piece which is still not downloaded to any higher layer piece. This means that, when the algorithm starts, there are as many items as layers for each group. The first item contains only the base layer piece, the second item the base and first enhancement layer pieces, etc., until the final item, which contains the pieces for all layers. All the items belonging to the same time slot form a group. Thus, we have to select at most one item from each group and for each time slot.

A further extension of the *MCKP* is the multiple-choice multi-dimensional knapsack problem (*MMKP*). In this case there are several knapsacks (neighbour peers), each of them with limited (download) capacity. The resource needs of the pieces can be described as a vector because the piece can be downloaded from a number of neighbour peers. The goal of applying the *MMKP* to our problem is to optimize the value of the selected pieces while none of the resources is exceeded. The main advantage of this approach is that it can consider the individual resources (bandwidth) provided by the neighbour peers instead of only the overall bandwidth.

The *MMKP* can be easily mapped to the piece and peer selection problems. Due to its performance and applicability the *HEU* algorithm presented in [7] was selected for implementation. Although the algorithm can deal well with the dependency between the layers, it does not consider the dependency between pieces in the subsequent timeslots (to avoid quality switches). Its complexity is $O(m^2 \cdot (n\text{-}1)^2 \cdot z)$, which already includes the peer selection process.

4.5 The Greedy Algorithm

Based on the already existing greedy algorithms for the *KP*, a greedy algorithm that specifically considers the requirements of the piece selection was developed. Before the algorithm starts, the pieces in the queue are ordered decreasingly according to their weighted utility. This sorting of the pieces in the queue has a complexity of $O(m \cdot n \cdot log(max(m,n)))$. The pseudo-code description of the piece selection algorithm is provided in Algorithm 1:

inputs: q (piece queue), bw (actual free bandwidth)
outputs: r (list of pieces to download)

1. for all pieces in q
2. if $c_j < bw$
3. add p_{ij} to r
4. update bw

Algorithm 1. Greedy Piece Selection Algorithm.

The algorithm selects as many pieces as possible (depending on the currently available free bandwidth) from the beginning of the piece queue. As the pieces are sorted according to their weighted utility, the most useful pieces are selected for download. In line 2, a check is performed if the cost of the actual piece is lower than the available bandwidth. The complexity of the greedy piece selection algorithm without the initial sorting is $O(m \cdot n)$, but the overall time complexity is $O(m \cdot n \cdot log(max(m,n)))$.

Although the greedy algorithm does not explicitly consider the dependencies between the pieces, it selects only useful pieces if condition (11) is fulfilled:

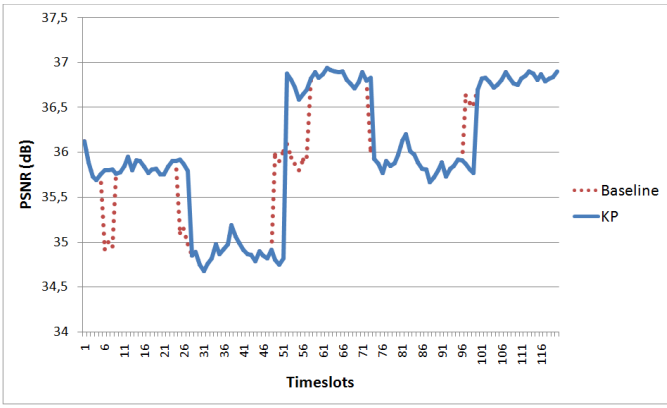

Figure 2. Evaluation of the Algorithms.

$$\left(t_{i'} \leq t_i \; and \; l_{j'} \leq l_j\right) \Rightarrow \left(wu_{i'j'k} \geq wu_{i\,j\,k}\right) \qquad (11)$$

Again, this condition is in conformance with the definition of the distortion reduction importance (d_j), which specifies that pieces of lower layers have a higher importance, and the utility (3), where fewer remaining timeslots result in a higher utility. Compared to condition (10), which ensures that the *DP* algorithm selects only useful pieces, condition (11) is less restrictive regarding the costs. It only implies that the utility per cost unit is higher for lower layer pieces, which should always be the case if the utility parameters are selected correctly.

5. EVALUATION OF THE ALGORITHMS

In this section the evaluation of the *KP*-based algorithms (the *DP*, *HEU*, and greedy algorithms) is presented. Firstly, the algorithms are compared to a simple layered piece-picking algorithm (the baseline algorithm), and the advantages of the *KP*-based algorithms in comparison to the baseline algorithm are presented. Additionally, the differences between the three *KP*-based algorithms are discussed.

The performance of the algorithms described in Section 3 was tested using the Oversim P2P simulation framework [8], which is based on the OMNeT++ simulation framework [9]. In order to support the protocols utilized within the *NextShare* P2P system and the piece-picking algorithms, we modified Oversim and implemented a new overlay (the *NextShare* protocol) and a number of new applications (the piece-picking algorithms). All algorithms were tested in a number of different settings where the following parameters were adjusted: number and bandwidth of the neighbour peers, number and cost of layers, sliding window size, network conditions, and the utility parameters.

The simulations have shown that the performance of the *KP*-based algorithms in terms of received video peak signal-to-noise ratio (PSNR) is very similar, as these algorithms use the same utility formula to assign priority to the pieces. Nevertheless, the algorithms differ in terms of time complexity (and hence runtime). Some additional differences between the *KP*-based algorithms are also discussed at the end of this section.

However, to illustrate the advantages of the *KP*-based algorithms, they are also compared to an efficient simple piece-picking algorithm, the baseline algorithm. The baseline algorithm works similar to the algorithm for the sliding window initialization. The algorithm considers all pieces within the sliding window that are not currently being downloaded and firstly selects the pieces from the lowest layer, starting with the earliest deadline, and then continues to select pieces from the next higher layer, and so on. Although this algorithm is rather simple to implement it achieves quite good results and is, thus, compared to the other algorithms presented in this paper.

In Figure 2 the received video quality for streaming a video sequence is illustrated as an example of the experiments with our simulation framework. The results for the algorithms discussed based on this test run have been similar over numerous test runs, but due to space constraints a single test run is presented. The video has a length of approx. five minutes (120 timeslots with 2.5 seconds each). It was encoded with three quality layers (500 kpbs, 800 kpbs, and 1000 kpbs) using medium-grain scalability (MGS). For the encoding process our optimized reference encoder [5] was used. The reason for using only quality layers and no spatial layers for the test sequence was to allow an easy comparison of the layers' PSNR values.

The peer for which the results are presented is connected to four neighbour peers. Every minute a change of the network conditions occurs. At the beginning, the download bandwidth provided by the neighbour peers allows to download the first two layers. After the first minute, the download bandwidth decreases to allow the download of only the base layer. After another minute, the available download bandwidth is increased to allow the download of all layers. With the start of the fourth minute the bandwidth decreases to allow the download of two layers and for the final minute the bandwidth increases to allow the download of all layers.

The figure shows the PSNR of the received video for each of the 120 timeslots at a single peer. The PSNR for a piece is determined by calculating the average of the PSNR values for all frames contained within the piece. The buffer initialization phase takes 5 timeslots and the sliding window size is 10 timeslots (~25 seconds). The figure illustrates the differences of the *KP*-based algorithms to the baseline algorithm. It should be noted to the performance of the *KP*-based algorithms can differ in specific cases (discussed later in this section), but for the settings of the test-run their performance in terms of received video PSNR was the same, as they use the same formula for utility calculation.

At the beginning of the streaming the network conditions allow to download the first two layers. The baseline algorithm performs an unnecessary quality switch at the beginning, as the buffer for the lower layers was only partly filled during the initialization phase (i.e., 5 of the 10 timeslots of the sliding window were filled) and the baseline algorithm firstly fills the entire sliding window for the lower layers before downloading the pieces for the desired quality. On the other hand, the *KP*-based algorithms take the more urgent deadline of the higher layer pieces into account and make sure that no quality switch is performed. At timeslot 25 the first decrease of the network bandwidth occurs. The baseline algorithm reduces the quality immediately, as it only filled the buffer for the lower layers. The *KP*-based algorithm can delay the quality switch for a few timeslots due to the higher layer buffer filling, which can be useful if there is just a short fluctuation of the network conditions.

When the quality increases to allow download of all layers at timeslot 49, the baseline algorithm increases the quality as soon as it has enough bandwidth to download one higher layer piece, while the *KP*-based algorithms fill the buffer and delay the step-up in quality due to the algorithm monitoring the quality switches

(again, this avoids quality switches when there are only temporary network fluctuations). However, the algorithms can subsequently perform a switch directly to the highest layer. At timeslot 73 the network bandwidth decreases to allow the download of two layers, and at timeslot 97 the network bandwidth changes again to allow the download of all layers. The behaviour of the algorithms is similar to the previous changes of the network bandwidth.

The main advantages of the *KP*-based algorithms in the presented test run are that unnecessary quality switch during initialization are avoided, as the consideration of the deadline during the utility calculation ensures that urgent pieces of higher layers are downloaded in time. Additionally, the *KP*-based algorithms react better to temporary bandwidth fluctuations due to the filling of the buffer also for higher layers (if the network conditions get worse) and the monitoring algorithm which avoids premature quality switches to higher layers (if the network conditions improve). In the presented test run the changes in bandwidth conditions remained always constant for a minute, but if, e.g., the decrease in bandwidth at timeslot 25 would only last for a few seconds, the baseline algorithm would perform a quality switch that the *KP*-based algorithms could avoid.

As the *KP*-based algorithms use the same formula for the utility calculation of the pieces, the results are the same for the test sequence in Figure 2. However, the *KP*-based algorithms differ in some aspects. Firstly, the complexity of the greedy algorithm that has been specifically developed for piece-picking is at $O(m \cdot n \cdot log(max(m,n)))$. The complexity of the *DP* algorithm is $O(S \cdot m \cdot n)$, or $O(S \cdot m \cdot n^2)$ if the dependency between the pieces is considered. Finally, the complexity of the *HEU* algorithm is $O(m^2 \cdot (n-1)^2 \cdot z)$, but includes the peer selection process as well. However, the peer selection process can be performed on its own with a complexity of $O(m \cdot n \cdot z)$, which makes the *HEU* algorithm still rather complex in comparison to the other algorithms. First runtime profiling results have confirmed what is already indicated by the time complexity statements, i.e., that the greedy algorithm can perform the piece selection significantly faster than the *DP* and the *HEU* algorithm. It should also be noted that the runtime performance of the *KP*-based algorithms depends strongly on the sliding window size (and hence the number of pieces that need to be considered for download).

In terms of received video PSNR the algorithms usually perform similar, but the *DP* algorithm without considering the dependency can download useless pieces, if the bitrate of the higher layer is smaller than the bitrate of the next lower layer (i.e., there is only sufficient bandwidth to download the higher layer piece, see (10)). For the greedy algorithm, the condition for selecting only useful pieces (11) is less restrictive and only requires that the utility per cost unit is higher for lower layer pieces.

6. CONCLUSION

In this paper a number of piece-picking algorithms based on the *KP* have been presented. These algorithms have been well investigated in combinatorial optimization for some time and find the solution for an optimization problem onto which the piece-picking problem can be mapped. We have shown that these algorithms can be applied to the piece-picking problem considering the requirements of a *Bittorrent*-based system. The greedy algorithm can perform as well as the other *KP*-based algorithms at clearly lower complexity. In the evaluation section, a comparison of the *KP*-based algorithms to a baseline algorithm that is often applied in the context of layered piece-picking has been presented, in which the *KP*-based algorithms have shown a better performance during initialization and when quality switches occur.

Due to space constraints the evaluation of the *KP*-based algorithms was limited to the presentation of a single test run that is representative for the results gathered during many experiments in our simulation framework. In the future, the proposed algorithms will be integrated into our *NextShare* P2P system and extensively tested in our project's living lab [10], and a more detailed evaluation will be performed.

7. ACKNOWLEDGMENTS

This work is supported in part by the European Commission in the context of the P2P-Next project (FP7-ICT-216217). Additional support of the Hungarian Science and Technology Foundation (AT-2/07), the Austrian Agency for International Cooperation in Education and Research (HU-6/08), and the Hungarian National Science Fund and the National Office for Research and Technology (Grant No. OTKA 67651) are gratefully acknowledged.

8. REFERENCES

[1] Schwarz, H., Marpe, D., and Wiegand, T. 2007. Overview of the Scalable Video Coding Extension of the H.264/AVC Standard. *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no. 9 (Sept. 2007), 1103-1120.

[2] Szkaliczki, T., Eberhard, M., Hellwagner, H., and Szobonya, L. 2010. Piece Selection Algorithm for Layered Video Streaming in P2P Networks. *Electronic Notes in Discrete Mathematics, Elsevier,* vol. 36, 1265-1272.

[3] Martello, S. and Toth, P. 1990. Knapsack Problems: Algorithms and Computer Implementation. John Wiley and Sons, New York.

[4] Liu, Z., Shen, Y., Ross, K. W., Panwar, S. S., and Wang, Y. 2009. LayerP2P: Using Layered Video Chunks in P2P Live Streaming. *IEEE Transactions on Multimedia*, vol. 11, no. 7 (August 2009), 1340-1352.

[5] Capovilla, N., Eberhard, M., Mignanti, S., Petrocco, R., and Vehkaperä, J. 2010. An Architecture for Distributing Scalable Content over Peer-to-Peer Networks. *Proceedings of the Second MMEDIA Conference*, 1-6.

[6] Andonov, R., Poirriez, V., and Rajopadhye, S. 2000. Unbounded Knapsack Problem: Dynamic Programming Revisited. *European Journal of Operational Research*, No. 123, Issue 2, 394-407.

[7] Khan, S., Li, K. F., Manning, E. G., and Akbar, M. M. 2002. Solving the Knapsack Problem for Adaptive Multimedia Systems. *Studia Informatica Universalis 2 (1)*, 161-182.

[8] Baumgart, I., Heep, B., and Krause, S. 2007. Oversim: A Flexible Overlay Network Simulation Framework. *Proceedings of the 10th IEEE Global Internet Symposium* (May 2007), 79-84.

[9] Varga A., and Hornig R. 2008. An Overview of the Omnet++ Simulation Environment. *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems*, 1-10.

[10] P2P-Next Living Lab, http://livinglab.eu, last accessed on 26/07/2010.