J Supercomput (2013) 66:700–720
DOI 10.1007/s11227-012-0858-7

Churn-aware optimal layer scheduling scheme for scalable video distribution in super-peer overlay networks

Yong-Hyuk Moon · Jeong-Nyeo Kim · Chan-Hyun Youn

Published online: 15 January 2013

Abstract To model a layered video streaming system in super-peer overlay networks that faces with heterogeneity and volatility of peers, we formulate a layer scheduling problem from understanding some constraints such as layer dependency, transmission rule, and bandwidth heterogeneity. To solve this problem, we propose a new layer scheduling algorithm using a real-coded messy genetic algorithm, providing a feasible solution with low complexity in decision. We also propose a peer-utility-based promotion algorithm that selects the most qualified neighbor to guarantee the sustained quality of streaming despite high intensity of churn. Simulation results show that the proposed layer scheduling scheme can achieve the most near-optimal solutions compared to the four conventional scheduling heuristics in the average streaming ratio. It also highly outperforms those with different peer selection strategies in terms of the average bandwidth (6.9 % higher at least) and the variation of utilization (11.3 % lower at least).

Y.-H. Moon (✉) · J.-N. Kim
Software Research Laboratory, Electronics and Telecommunications Research Institute (ETRI), Daejeon, South Korea
e-mail: yhmoon@etri.re.kr

J.-N. Kim
e-mail: jnkim@etri.re.kr

Y.-H. Moon
Department of Information and Communications Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea
e-mail: yhmoon@kaist.ac.kr

C.-H. Youn
Department of Electrical Engineering, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, South Korea
e-mail: chyoun@kaist.ac.kr

Keywords Content delivery · Layer-coded video · Streaming · Churn resilience · Peer-to-peer network · Genetic algorithm

1 Introduction

Currently, scalable video distribution services using peer-to-peer (P2P) networks have been rapidly adopted for various purposes to achieve fast and reliable dissemination of massive data from multiple sources. However, the presence of multiple supplying peers requires a new solution of the video segment allocation problem under heterogeneous bandwidth constraints [1]. In particular, video streaming over volatile or harsh environments in which nodes are prone to departure (i.e., leave or fail) becomes further complicated [2, 3].

There exist several studies in the literature to address the streaming control and the node stability issues in P2P networks. Block scheduling schemes in P2P networks have been intensively studied. In the early stage, a random and a local rarest first (LRF) scheduling have been employed in Chainsaw [4] and in DONet [5], respectively. Moreover, PALS [6] has been proposed for the adaptive streaming of layered-coded video and focuses on coping with bandwidth variations. However, its evaluation has not been performed on mesh overlay and a new sender is randomly selected when some layers are lost or delayed. In [7], DONLE has formulated a priority based block scheduling problem to maximize the average throughput of data-driven P2P streaming. Although peers are very unstable in terms of session time, resulting in the significant deterioration of streaming quality, loss recovery has been little discussed in these works. Conventionally, neighbor selection strategies focusing on different criteria (e.g., propagation delay and bandwidth capacity) have been proposed as well in order to ensure the equable allocation of aggregated bandwidth from multiple peers. Li Xiao has reported that the amount of workload on an ordinary-peer is directly related to deciding the optimal size and stability of the super-peers in [8]. As discussed in [9], assuming that super-peer overlay network is modeled by a bimodal random graph, the stability of networks entirely depends on the fraction of super-peers. However, the more super-peers do not always guarantee better performance (e.g., throughput) as discussed in [6]. Further, these studies have paid less attention to the real-time property of streaming service, even though their analytical model can offer a generic aspect of super-peer's churn on the network stability.

From the studies discussed, the main challenges addressed in this paper can be summarized as follows:

- Lack of Control. Conventional approaches based on the server-based content delivery system turn out to be insufficient due to its large operation cost (e.g., server overload) and the existing block scheduling schemes also face with the lack of adaptive control for differentiated streaming quality. Hence, scaling for streaming system is still painful [10].
- Dynamic Volatility. Random churn causes the shortage of collective availability (i.e., aggregated bandwidth), resulting in that streaming performance cannot be guaranteed within a required service level. Confidently, consuming peers suffer from poor fault tolerance.

To tackle the dynamic churn [2] of multiple supplying peers, first we exploit the super-peer overlay structure [11], which highly ensures collective availability for large-scale streaming. To use advantages of layered coding (i.e., scalable video coding [12]), we also adopt a scalable (multilayered) video[1] for streaming. This approach is very efficient to provide the differentiated streaming quality by adaptively extracting only necessary bit-streams of layers according to the given bandwidth constraints.

Despite the potential benefits of combining the two techniques, finding an optimal layer allocation generally requires the super-exponential computation complexity (i.e., NP-complete) under heterogeneous bandwidth. Further, an optimal solution may not be always achieved due to the fast converging to local optima in the existing heuristics [4–7]. In addition to that, the conventional approaches which are designed to reduce the adverse effects of node departures in P2P file sharing or task execution [8, 9] are significantly inadequate to handle the churn-vulnerable streaming service; namely, those may well achieve good performance, only if the P2P networks can be stably maintained.

In this paper, we propose a churn-aware optimal layer scheduling scheme which consists of two algorithms: (1) a real-coded messy genetic algorithm (rmGA) [13] based layer scheduling (GALS), which provides a near-optimal layer allocation with acceptable time complexity; and (2) peer-utility-based promotion (PUP), which updates a group of corresponding super-peers by adding new neighbors for rapidly compensating the layer losses.

The primary contributions of this paper are as follows:

- We formulate a layer scheduling problem from understanding some constraints which originate from the layer dependency, the transmission rule, and the bandwidth heterogeneity in P2P networks. Further, we suggest how to prioritize each layer in order for supporting differentiated streaming services.
- To provide high adaptability for the layer control, we propose a novel layer scheduling algorithm using a real-coded messy genetic algorithm that is locally performed at each receiving peer.
- Also, a proposed peer-utility-based promotion guarantees the dynamic reconfiguration of layer allocation under the random churn model, so that horizontal scalability of all peers can be achieved.
- Simulations results demonstrate superiority of the proposed layer scheduling scheme in comparison with the conventional layer scheduling heuristics and neighbor selection approaches.

The remainder of this paper is organized as follows. Section 2 describes the layered streaming model in P2P networks with respect to its construction, transmission, and download rate. In Sect. 3, we formulate a layer scheduling problem and then derive an optimal policy for this problem. We propose a churn-aware optimal layer scheduling scheme and also discuss behavior of dynamic churn in Sect. 4. Performance of the

[1]Unlike the conventional video compression algorithms based on single-layer approaches, such as MPEG-4 and H.264/AVC, the scalable video is variable and flexible according to various network and device conditions. It can be dynamically retransformed in terms of spatial, temporal, and quality scalabilities.

proposed layer scheduling scheme is evaluated and simulation results are discussed in Sect. 5. Finally, we conclude this paper in Sect. 6.

2 Layered streaming model in P2P networks

A layered streaming model based on the two-level hierarchy [14], comprising ordinary-peers and super-peers allows decentralized networks to run more efficiently by exploiting heterogeneity and distributing load to peers that can handle the burden.

We first adopt the layered-coding technique in order to ensure the inexpensive and adaptive data (i.e., layer) dissemination in P2P overlay networks. A scalable video is accumulatively constructed with a set of layers (i.e., a sub-stream) $L = \{l_1, l_2, \ldots, l_v\}$ by the layered coding, where l_1 is a basement layer and others are enhancement layers (see Fig. 1). In particular, the enhancement layers are applied to improve streaming quality. In general, the overall streaming quality may be directly proportional to the number of received layers at a consuming peer.

Definition 1 (Layer dependency) To play a specific scene of the scalable video, a lower layer is essentially required because a higher layer is not independently decodable without a lower one [12].

Furthermore, we consider Layered Streaming in Super-peer Overlay Networks (LSON) using the gossiping protocol [8] for encouraging interaction among neighbor peers by which the peer partnership is established. As illustrated in Fig. 2, most peers are ordinary-peers $O = \{O_x \mid x = 1, 2, \ldots, n\}$ of consuming bit-streams, while a small fraction of peers act as super-peers for coordinately supplying layers. However, the roles of peers are neither exclusive nor prescribed.

Definition 2 (Service bootstrapping) After such a streaming service on LSON is activated for a particular V_{LC}, a streaming server Π sends the layered streams to the bootstrapping peers B by priority for enabling the distributed cache effects of layers.

Definition 3 (Peer partnership) When an ordinary-peer x, O_x, newly joins LSON to receive layered video streams, it first gets a list of randomly selected super-peers,

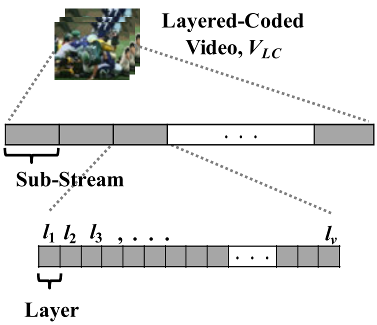

Fig. 1 Structure of a layered-coded video (V_{LC})

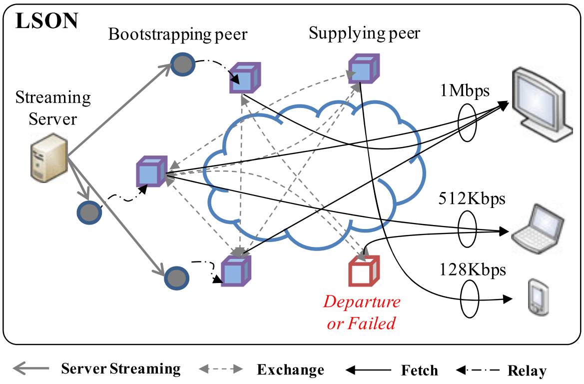

Fig. 2 Conceptual diagram of LSON

$S = \{S_i \mid i = 1, 2, \ldots, m\}$ and Π from the nearest bootstrapping peer for reducing startup latency.

Using the aforementioned partnership, each peer broadcasts its layer availability to other peers, so that a layer scheduler at each peer can aggregate this information. A scalable video originates from Π and each peer receives some layers of the scalable video from supplying peers such as Π, S, and O. Simultaneously, each peer can supply partial or whole layers received or stored.

For the sake of simplicity, we assume that all of the super-peers and ordinary-peers have the same upload rate u_s and u_o, respectively, where $u_s > u_o$. Let $r^+(t)$ $(t \geq 0)$ denote aggregated download rate from those peers at time t.

Property 1 (Bound of $r^+(t)$) We note that $r^+(t)$ must be bounded by the download link capacity d_c of a peer, that is,

$$r^+(t) \leq d_c, \quad \forall t \geq 0. \tag{1}$$

Let $\varphi_s(t)$ and $\varphi_o(t)$ $(t \geq 0)$ denote $|S|$ and $|O|$, respectively, for time t. We also assume that $|O|$ is proportional to $|S|$ as discussed in [8]; that is, $\varphi_o(t) = \alpha \cdot \varphi_s(t)$, where α is some positive number. Thus, $r^+(t)$ of Π, S, and O in LSON is equal to $u_\Pi + (u_s + \alpha \cdot u_o) \cdot \varphi_s(t)$, where u_Π is a upload rate of Π.

Property 2 (Share of $r^+(t)$) Since all peers share $r^+(t)$, the upload rate available for each O_x, $\forall x \in X$ $(X = \{1, 2, \ldots, n\})$ is given by

$$\frac{u_\Pi + (u_s + \alpha \cdot u_o) \cdot \varphi_s(t)}{(1 + \alpha) \cdot \varphi_s(t)}. \tag{2}$$

From Eqs. (1) and (2), $r^+(t)$ can be determined as follows:

$$r^+(t) = \min\left\{d_c, \frac{u_\Pi + (u_s + \alpha \cdot u_o) \cdot \varphi_s(t)}{(1 + \alpha) \cdot \varphi_s(t)}\right\}. \tag{3}$$

However, $\varphi_s(t)$ varies over time due to the departures and failures, so that $r^+(t)$ can be fluctuated unexpectedly. This fact implies that some peers possibly suffer from layer missing, which causes the performance degradation [15]. One crucial restriction is that although each peer has heterogeneous bandwidth capacity in general, the above model does not consider how to offer differentiated streaming quality to peers according to their download rates.

Therefore, we will propose how a layer allocation (i.e., schedule) is adaptively generated to fully utilize the given $r^+(t)$ by coping with the fluctuation of $r^+(t)$ and peer heterogeneity in Sects. 3 and 4, respectively.

3 Problem formulation for LSON

3.1 Layer streaming process

For ease of explanation of layer streaming process, we present Fig. 3 that illustrates the overall process of optimal layer allocation in LSON, which comprises four phases such as availability aggregation, layer scheduling, layer fetching, and peer promotion.

Availability aggregation As previously discussed, a layer scheduler at each O_x can aggregate the layer availability information of S for a particular V_{LC}:

$$A_v^x = \left\{a_{i,j}^x \in \{0, 1\} \mid 1 \leq i \leq m,\ 1 \leq j \leq v\right\},$$

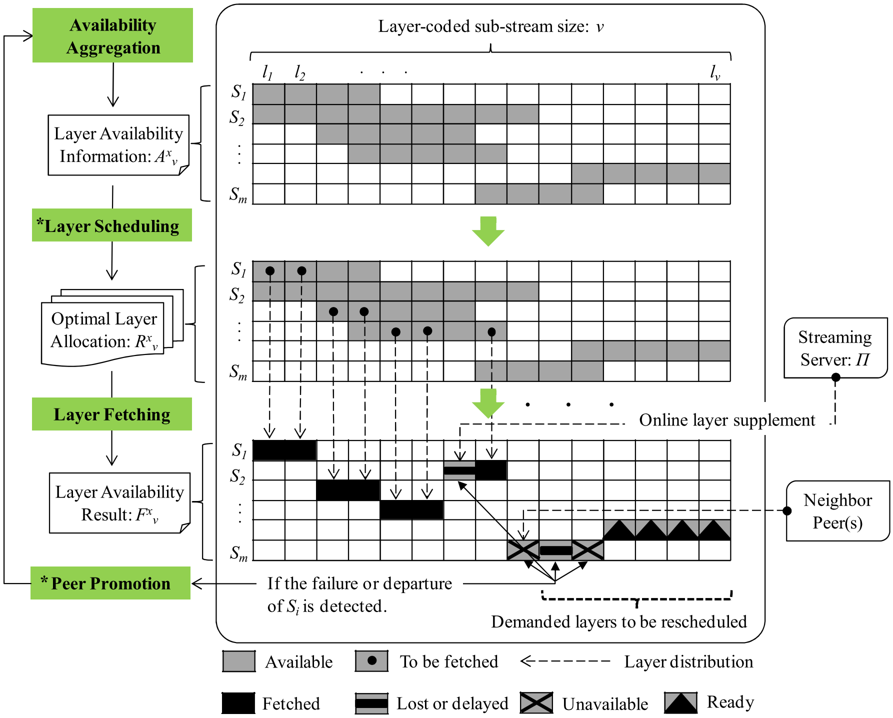

Fig. 3 Overall process of optimal layer allocation in LSON; The two phases marked by an *asterisk* are performed by proposed individual algorithms whose details will be discussed in Sect. 4

where $a^x_{i,j} = 1$ indicates that a super-peer i, S_i, can offer a layer j, l_j, that is received from Π, B, or other peers and $a^x_{i,j} = 0$ otherwise. As depicted in Fig. 3, an upper matrix represents A^x_v given to O_x, where a gray-colored cell denotes a layer that is available at a particular super-peer. Thus, a layer scheduler can identify how many layers can be available at each S_i by counting ones along with ith row of A^x_v as $\sum_{j=1..v} a^x_{i,j}$.

Layer scheduling Using the given A^x_v, a layer scheduler of O_x can initially decide an optimal layer allocation (referred to as schedule or decision), that is,

$$R^x_v = \left\{ r^x_{i,j} \in \{0, 1\} \mid 1 \le i \le m,\ 1 \le j \le v \right\},$$

where R^x_v indicates which layer would be fetched from which peer. $r^x_{i,j} = 1$ means that O_x would receive l_j from a super- or ordinary-peer i; on the other hand, $r^x_{i,j} = 0$ means that l_j would not be scheduled on a peer i. Thus, R^x_v can be considered as a set of pointers that are depicted by a black dot on the gray-colored cells of a middle matrix denoted by R^x_v in Fig. 3.

Layer fetching By referring to the initialized decision, R^x_v, O_x can preferentially begin fetching relevant layers from S in parallel, where an already fetched layer is represented by black-colored cells in the layer allocation result denoted by $F^x_v = \{f^x_{i,j}\}_{m\times v}$. However, some layers can exceptionally be lost or delayed (denoted by gray-colored cells with a black bar) at any time due to the lack of outgoing bandwidth or the departure of a specific S_i. Moreover, the layer can be unavailable (denoted by gray-colored cells with a black x-bar) for a while since gaining exclusive access to layers on S is generally granted to only one O_x at a time.

For mitigating the impact of these undesirable cases, (1) missing (lost, delayed, or unavailable) layers can be promptly sent by Π to facilitate the maximum use of O_x's incoming bandwidth; or (2) those layers might be individually forwarded from other ordinary-peers (neighbors). Of the two countermeasures, the former will reach to the limit because Π has to consume much outbound bandwidth and then it will finally encounter difficulties to handle these challenging situations. Since the peer partnership of LSON is established using the mesh-based P2P paradigm, the latter would be somewhat helpful to supplement the insufficient availability of super-peers. Nevertheless, discovering relevant layers that may be very sparsely distributed among random peers is both inefficient and rigid to perform a fast recovery in some cases, resulting in that the layer request and fetching operations work as random scheduling. Therefore, it is hard to achieve the optimal scheduling performance.

Peer promotion To overcome these problems, those layers should be rescheduled with an updated group of super-peers that are adaptively reconfigured by a peer promotion process when the failure or departure of S_i is detected (see Fig. 3). Rather than attempting to use the full bandwidth capacity of Π or depending on the individual and random assistance of other ordinary-peers, this process can provide higher adaptability to minimize the loss of collective layer availability if O_x's peer partnership is assumed to be well established in LSON. This is because the promotion process aims to rapidly reconfigure a set of super-peers of O_x by adding the most capable peer that has sufficient availability to supplement missing or ready (denoted by gray-colored cells with a black triangle in Fig. 3) layers. Through this process, a layer scheduling process can decide a new R_v^x for re-fetching those layers by reflecting the availability information of an added super-peer with the currently updated A_v^x of the existing (stable) super-peers together.

However, the two static approaches discussed above can be alternatively used before a rescheduling process that is composed of selection of a new super-peer, update of A_v^x, and generation of a new R_v^x is initiated by a peer promotion process. In this case, some layers that are already supplemented by these approaches will not be included in the rescheduling process in order to avoid duplication of effort in re-fetching those layers. Therefore, ordinary-peers complementarily enhance their streaming quality by exchanging and relaying different layers of V_{LC} under such a dynamic mesh overlay comprising the corresponding super-peers and Π.

The layer streaming process comprising the above four phases iteratively proceeds until the end of sub-stream of V_{LC}. In the aforementioned model, we can guarantee optimal layer allocation and its rapid rescheduling even though some layers are unpredictably missed due to random churn.

3.2 Optimal policy for layer scheduling

In the LSON model, we assume that $r^+(t)$ (i.e., network bandwidth) is normalized as the number of layers that each peer can send or receive. Hence, the total bandwidth consumed at O_x (denoted by Q_x^T) is calculated by summing the number of layers fetched from Π, S, and O (other than O_x), respectively, as follows:

$$Q_x^T = \sum_{i\in I}\sum_{j\in J} r_{i,j}^x + Q_x^{\Pi}, \quad I = \{1, 2, \dots, m\} \cup \big(X - \{x\}\big),\ J = \{1, 2, \dots, v\}, \tag{4}$$

where Q_x^{Π} represents the upload rate available from Π to O_x.

Since our objective is not only to save the cost of Π but also to maximize the effectiveness of P2P data dissemination for $\forall x \in X$, an optimal policy of the layer scheduling can be formulated as follows:

$$\text{Max:} \quad \sum_{x\in X}\big(Q_x^T - Q_x^{\Pi}\big) = \sum_{x\in X}\sum_{i\in I}\sum_{j\in J} r_{i,j}^x, \tag{5}$$

where the five constraints must be ensured in terms of layer dependency, transmission rule, and bandwidth (rate) heterogeneity, when a layer scheduler of O_x attempts to make a new decision for layer allocation.

- (C1) $r_{i,j}^x \le a_{i,j}^x, \forall i \in I, \forall j \in J, \forall x \in X : l_j$ would be fetched if and only if the layer is available at S_i.
- (C2) $\sum_{i\in I} r_{i,j}^x \ge \sum_{i\in I} r_{i,j+1}^x, \forall j \in J, \forall x \in X$: Due to the layer dependency, the number of received l_j is larger than or equal to that of received l_{j+1} for all layers.
- (C3) $\sum_{i\in I} r_{i,j}^x \le 1, \forall j \in J, \forall x \in X$: Once an arbitrary layer is fetched from S_i, a duplicated request is limited.

- (C4) $\sum_{j \in J} r^x_{i,j} \le u_{i,x}, \forall i \in I, \forall x \in X$: A total number of layers received from only S_i is less than or equal to the outgoing bandwidth capacity allocated from S_i to O_x (denoted as $u_{i,x}$).
- (C5) $Q^T_x \le r^+(t), \forall x \in X : Q^T_x$ cannot exceed $r^+(t)$.

Thus, a greater value of Eq. (5) guarantees a better scheduling solution in terms of the total consumed bandwidth (i.e., fetched layers). In order to achieve an optimal layer allocation, we formulate a layer scheduling problem with the three types of constraint discussed. However, solving this problem for all consuming peers is impractical (i.e., global optimization); since, this approach highly increases the problem size. If local availability information is sufficiently updated for each peer, a local layer scheduler could provide a feasible or near-optimal decision for scalable video distribution. Therefore, Eq. (5) needs to be rewritten as Max: $\sum_{i \in I} \sum_{j \in J} r^x_{i,j}$ in order for adopting the receiver-driven layer scheduling approach.

Further, we only consider how to assess Q^T_x for an initial layer allocation in Eq. (4), where all layers are scheduled if O_x requests full-rate streaming of V_{LC}; however, only missing or ready layers become an object of attention in rescheduling as mentioned earlier. So, each case will need to be individually considered as follows:

$$J = \begin{cases} \{1, 2, \ldots, v\}, & \text{if initial scheduling,} \\ L_{\mathrm{mis}} \cup L_{\mathrm{ready}}, & \text{if rescheduling.} \end{cases} \tag{6}$$

In Eq. (6), L_{mis} and L_{ready} are a set of missing layers and ready (not fetched yet) layers, respectively. Therefore, a set of demanded layers are set by the union of the two sets in case of rescheduling. More specifically, Q^T_x can be obtained by summing the total number of layers fetched by individual scheduling cases.

3.3 Service level decision

If higher layers are aggressively requested, the fetch request for lower ones possibly would not be able to be served before their playback deadline. To solve this problem, we assign the same (highest) importance to lower layers in order to guarantee the standard quality level of streaming services; on the other hand, lower priority is given to higher layers. The above consideration gives us a good motivation to provide differentiated streaming quality in LSON. As summarized below, we apply a different prioritization rule for the two request types. Let l^x_{th} denote a threshold layer for O_x.

(i) The requests for layer l_1 to l^x_{th}, is considered to be of the highest priority. Since we assume that the same (highest) priority is set to all those layers, we do not prioritize those layers.
(ii) For layer l^x_{th+1} to l_v, lower layers have higher priorities than those of higher layers. For every layer, a request is iteratively sent to supplying peers in order of importance.

For this reason, a layer scheduler should decide a threshold layer that is used to classify L into two classes: lower layers and higher layers. Let L_x and L_m denote a set of layers to be scheduled at O_x and a minimum set of layers to be fetched at O_x, respectively.

Lemma 1 (Threshold layer) *Suppose that L_m is given by Eq.* (7), *we can decide l^x_{th}.*

$$|L_m| = \begin{cases} |L_x| - |L_{\text{mis}}| < r^+(t), & \text{if missing layers are found,} \\ |L_x| + |L_{\text{sur}}| \le v, & \text{if surplus layers are found,} \end{cases} \tag{7}$$

where L_{mis} and L_{sur} are a set of missing layers and surplus layers, respectively ($L_{\text{mis}}, L_{\text{sur}} \subset L$).

Proof If aggregated download rate is equal to or larger than the size of whole layers L (i.e., $r^+(t) \ge v$) then full-rate streaming quality would be supported; in other words, L_x is equal to L ($L_x \leftarrow L$). Otherwise ($r^+(t) < v$), $L_x \leftarrow r^+(t)$. However, L_x could be reduced (increased) by the number of missing layers (the surplus upload rate of Π, S, and O_y, where $x \neq y$, $\forall x, y \in X$) in streaming. Thus, the minimum number of layers to be fetched at O_x, $|L_m|$, is intuitively determined as Eq. (7). Since l^x_{th} is the highest layer in L_m, an index of l^x_{th} can be given by $th \leftarrow |L_m|$, where th is a layer index of l^x_{th}. □

One definite advantage of using the threshold layer based fetching technique is that O_x can avoid excessive efforts (e.g., competition) for receiving layers l^x_{th+1} to l_v. By focusing on layers l_1 to l^x_{th} first, faster and more reliable data dissemination would be achieved in terms of O_x's $|L_m|$ from the different prioritization rule rather than dealing with all layers equally. In particular, this design principle is effective in implementing differentiated streaming service.

So far, we consider that maximizing the total number of received layers is an objective function for LSON. However, each layer has a different importance value as discussed. Besides, Definition 1 implies that in P2P networks the overall quality of scalable video streaming is not proportional to the amount of correctly received layers. To incorporate the concept of threshold layer into the modified equation of Eq. (5), Max: $\sum_{i\in I}\sum_{j\in J} r^x_{i,j}$, we reformulate an optimal policy for the quality guaranteed layer scheduling as follows:

$$\text{Max:} \quad \sum_{i\in I}\sum_{j\in J} r^x_{i,j} w^x_j, \tag{8}$$

where ω^x_j denotes a given priority to l_j. Thus, we have $\omega^x_1 = \omega^x_2 = \cdots = \omega^x_{th}$ and $\omega^x_{th+1} > \omega^x_{th+2} > \cdots > \omega^x_v$, where $\omega^x_j, \forall j \in J$ is assumed to be bounded between 1 and an arbitrary integer of more than 1. This weighted equation is much suitable to evaluate the differentiated streaming quality in LSON.

From the above discussion, the two important requirements can be derived as follows:

- (R1) To achieve an optimal layer allocation without violating the constraints (C1)–(C5) is a combinatorial optimization problem. Due to its large-scale search space, this problem also has been proved to be NP-complete in the strong sense [1]. So, a new scheduling heuristic is necessary.
- (R2) In addition to that, when churn occurs, a layer scheduler should reconfigure an initial allocation by selecting new super-peers in order to mitigate the performance degradation due to large propagation delays or high layer loss rates.

Next, we will discuss details of a new layer allocation scheme designed for LSON.

4 Churn-aware optimal layer scheduling scheme

The purpose of a churn-aware optimal layer scheduling scheme is to consistently provide a feasible solution during streaming process in LSON. In this section, we first describe how the proposed GALS algorithm rapidly reaches to a near-optimal solution while keeping time complexity low. Then, we suggest a new churn model for LSON and also propose a PUP algorithm for dynamic reconfiguration of initial allocation, when churn occurs.

4.1 rmGA based layer scheduling algorithm

rmGA offers the stochastic search ability to evolve an initial population of candidate schedules (called chromosomes) into an optimal one by recombination operations under the principle of the survival of the fittest. To codify an individual layer allocation to the scheduling problem on LSON, we first define a chromosome as a set of 2-tuples (called a gene) such as (super-peer index i, layer index j). For example, a chromosome u, $c_u = \{(3,1);(2,2);(4,3);(2,4)\}$, means that four layers can be separately fetched from S_3, S_2, S_4, and S_2 in a parallel manner. c_u is then simply translated to R_v^x, where r_{31}^x, r_{22}^x, r_{43}^x, and r_{24}^x are only equal to 1. Namely, c_u is very tractable to express various combinations of genes and is also compatible with R_v^x in this one-dimensional representation.

The following describes the rmGA search operations (see Algorithm 1).

- Step 1 (Initialization) Based on the above encoding schema, rmGA starts by initializing a population P_g $(g = 1)$ that comprises a fixed number of c_u, $1 \leq u \leq U$, where U is a fixed integer number.

Algorithm 1 rmGA based layer scheduling

Input: $A_v^x = \{a_{i,j}^x\}_{|I|\times|J|}$, $R_v^x = \{r_{i,j}^x\}_{|I|\times|J|}$, $L \in V_{\text{LC}}$
  $O_x \in O$, $S_i \in S$, Π, $g = 1$
Generate P_g for L by referring A_v^x;
while $g < g_{\max}$ **do** // $g_{\max}$: the maximum number of generations
  Evaluate $\forall c_u \in P_g, 1 \leq u \leq U$ by Eq. (8);
  if $\exists c_u \in P_g$ satisfying that $f_u \geq f_{th}$ **then**
    $c^{\text{opt}} = c_u$ and break;
  end if
  $(c_a, c_b) =$ Select parents by $Pr_{\text{sel}}(c_u)$;
  $c_o =$ Cut-and-Splices(c_a, c_q);
  $\hat{c}_o =$ Mutate$(c_o)$and then add $\hat{c}_o$ inP_{g++};
end while
Output: Translate c^{opt} into R_v^x;
O_x starts to fetch layers according to R_v^x;

- Step 2 (Evaluation) To evaluate a fitness value of each c_u, f_u, Eq. (8) is used. If f_u is sufficiently large enough compared to a predefined threshold of f_u, f_{th}, then c_u becomes an optimal layer allocation c^{OPT} and this procedure is finally terminated.
- Step 3 (Selection) Otherwise, chromosomes having relatively higher fitness values should be selected by such probability $Pr_{\mathrm{sel}}(c_u) = (f_u / \sum_{u=1..U} f_u)$ in the roulette-wheel selection [16] in order to choose superior solutions, which would be evolved in the next generation.
- Step 4 (Recombination) To obtain globally evolved solutions, cut-and-splices uniformly swaps two genes between c_a and c_b, where $1 \le a, b \le U$ and $a \neq b$, resulting in that c_o is produced. To explore local solutions, mutation is then performed by randomly exchanging two genes within each c_o. After these two operations, we can have more feasible offspring $\hat{c}_o$ in the next generation P_{g++}.
- Step 5 (Termination) All steps are iterated until the stopping condition ($g \ge g_{\max}$) is satisfied.

We note that the above operations must abide with (C1)–(C5). The more detailed review on genetic algorithm operations is discussed in [13, 16], and [17]. With rmGA, the proposed layer scheduling algorithm generates the optimal R_v^x, which maximizes the overall performance of layered streaming, while maintaining the search complexity low.

Definition 4 (Optimal instance size) In Algorithm 1, the size of the given problem instance, $\acute{I}$, is equal to the length of c_u, so that its size is determined by $|J|$. When rmGA reaches to c^{opt} with arbitrary $\acute{I}$ in the best possible convergence time, we say $\acute{I}$ is the optimal instance size, $\acute{I}^{\mathrm{opt}}$. In other words, rmGA applied to $\acute{I}^{\mathrm{opt}}$ can guarantee faster convergence time to near-optimal solutions compared with other sizes.

Lemma 2 (Complexity analysis of Algorithm 1) *Suppose that $\acute{I}^{\mathrm{opt}}$ can be determined, the computational complexity of rmGA on the layer scheduling problem is linearly increased despite the exponential growth rate Δs of the search space.*

Proof Computational complexity is a property of a problem and not an algorithm. Since v layers are iteratively streamed to O_x and each has different playback deadline, unlike a general class of optimization problem, a layer scheduler does not need to handle all layers at the same time. So, the problem size is divisible according to the given conditions on LSON. Moreover, the size of search space ${}_{|I|+|J|-1}C_{|J|}$ on the problem is an exponential function of both $|I|$ and $|J|$; however, Δs, is only determined by $|I|$, if $\acute{I}^{\mathrm{opt}}$ is known. As theoretically reviewed in [18], if $\acute{I}^{\mathrm{opt}}$ is given prior to layer scheduling, then the length of c_u is fixed; thus, $g_{\max}$ required to find c^{opt} is linearly increased (some constant c times $O(\acute{I}^{\mathrm{opt}})$) even though Δs exponentially increases. □

The superiority of the GALS algorithm can be demonstrated, merely depending on how much optimal solution is generated for each scheduling request when particular conditions (i.e., super-peers and demanded layers) are given. However, such conditions vary over time due to the volatility of peers. From the layer scheduling viewpoint, next we will discuss how the time-varying conditions can be managed for ensuring consistent provisioning of collective availability in harsh environments.

4.2 Dynamic churn model

Before discussing an adaptive neighbor selection method, we start by describing a dynamic churn model. As observed in [19], we model the dynamic behavior of peers in LSON as a regenerative *on/off* process $\{Y_i(t)\}$ for each S_i, where $Y_i(t) = 1$ if S_i is *on* at time t and 0 otherwise. So, a type of each S_i can be classified by distinct pairs of cumulative distribution functions (CDFs), defining durations of online and offline periods as $T := \{(F_1(t), G_1(t)), \ldots, (F_£(t), G_£(t))\}$ according to its degree of volatility. In this model, the heterogeneity of peers can be adjusted by the diversity factor £.

Lemma 3 (Percentage of remaining peers). *Let $E[\Phi_i]$ denote the expected lifetime of S_i. Given $E[\Phi_i]$, we can estimate a churn rate of each S_i from some arbitrary distribution as described in Eq.* (9).

$$H_i(t) = \left(\left(E[\Phi_i]\right)^{-1} \int_0^x \left(1 - F_i(z)\right) dz\right). \tag{9}$$

Proof Since $Y_i(t)$ is associated with a pair of $(F_i(t), G_i(t))$, one of these CDFs is independently and uniformly selected from T for each S_i. Let $R_i(t)$ denote the remaining lifetime of S_i at time t in the current *on* cycle. Assuming that LSON is sufficiently large and stationary, the CDF of $R_i(t)$ is defined by $H_i(t) := Pr[R_i(t) \le x | Y_i(t) = 1]$, $t \ge 0$, where it can be obtained as Eq. (9). □

Therefore, as a discrete event timer for churn, our dynamic churn model will be used for determining what percentage of peers should be left at each instance in time. Although the exponential distribution is typically used to model some phenomena resulting from a large number of independent events, it is not proper to model churn behavior because peer arrivals and departures are not completely independent as discussed in [20].

4.3 Peer-utility-based promotion

We suppose that each O_x is able to manage a set of current neighbors N_x, which potentially supply some demanded layers L_{dem} to O_x. To quantify the promotion criterion, rather than concentrating on one specific aspect for assessing eligibility of each neighbor, we consider three representative factors such as (1) a hop count $h_{k,x}$ between $\forall k \in N_x$ and O_x, (2) the upload bandwidth capacity allocated from k to O_x, $u_{k,x}$, and (3) the expected stability level of kth neighbor, σ_k. To make it possible, we need to transform the three different quantities into the same domain in the form of combined function.

Definition 5 (Utility function) The three factors are incorporated into a utility function $U(\cdot)$, where a peer utility of kth neighbor is defined as $U(k) := ((1/h_{k,x}) \cdot u_{k,x} \cdot \sigma_k)$.

Algorithm 2 Peer-utility-based promotion

if $E[R_i(t)] = 0 \parallel |L_{\text{mis}} \cup L_{\text{ready}}| > \tilde{n}$ // $\tilde{n}$: the maximum tolerable number of layers
  Set $L_{\text{dem}} = L_{\text{mis}} \cup L_{\text{ready}}$ // L_{dem}: a set of demanded layers for rescheduling
  while $\exists k \in N^x$ **do**
    Calculate $U(k)$ for $k \in N^x$
  end while
  Sort $\forall k \in N^x$ by the value of $U(k)$
  $S \leftarrow$ Promote k with the highest utility value of $U(k)$
  Set $\acute{S} = S \cup k$
  Call the *GALS algorithm* with $\acute{S}$ and L_{dem} // $\acute{S}$: a updated super-peer group of O_x
end if

Property 3 (Distance) A neighbor with fewer hops (from O_x to kth neighbor) is preferred due to the relatively low forwarding latency. So, abnormal delays would be prevented [21].

Property 4 (Layer availability) $u_{k,x}$ is calculated as $\sum_{j \in L_{\text{dem}}} a^x_{k,j}, a^x_{k,j} \in \{0, 1\}$, where $a^x_{k,j} = 1$ indicates l_j is available at kth neighbor and otherwise ($a^x_{k,j} = 0$), k cannot provide l_j.

Property 5 (Stability level) σ_k can be estimated as the number of connections sustained with other peers during its remaining session time. Thus, σ_k can be derived as '1 − an isolation probability of kth neighbor with randomly assigned degree of connectivity c_k', where an isolation occurs at the time that all links of peer is disconnected.

$$\sigma_k = 1 - \frac{\rho \cdot c_k}{(1+\rho)^{c_k} - 1} = 1 - \left(2c_k \cdot \frac{E[R_k]}{\delta + 2d}\right)\left(\left(1 + \frac{2E[R_k]}{\delta + 2d}\right)^{c_k} - 1\right). \quad (10)$$

In Eq. (10), ρ is a ratio of the expected remaining lifetime of kth neighbor, $E[R_k]$, to the expected search delay $E[D_k]$ of detecting its link disconnection. If we assume that O_x can detect the neighbor isolation through some transport layer protocol (i.e., keep-alive mechanism), $E[D_k]$ would be determined within $(0, (\delta + 2d)/2]$, where δ and d denote the keep-alive timeout and the search delay, respectively.

Algorithm 2 describes the detailed steps for discovering and promoting the most qualified neighbor(s) prior to rescheduling the demanded layers as below:

- Step 1: If the expected residual lifetime of S_i, $E[R_i(t)]$ is expired over the churn prediction model or some L_{mis} and/or L_{ready} are found, then departure of S_i or lack of collective layer availability can be detected. In this case, the PUP algorithm is activated for the neighbor discovery.

- Step 2: Then, layers required to be re-fetched, L_{dem}, are set by a union of L_{mis} and L_{ready}.
- Step 3: Individual values of $U(k)$ are calculated for $\forall k \in N_x$ by using a utility function and then each neighbor is ranked by its value.
- Step 4: Some neighbor(s) with the highest utility value is (are) selected and then added to an existing set of super-peers of O_x, S, for obtaining an updated set of super-peers, $\acute{S}$.
- Step 5: After O_x's S is reconfigured, a *GALS* algorithm is re-executed with $\acute{S}$ and L_{dem}. Through the above process, loss of L_{mis} and/or L_{ready} can be adaptively recovered.

By assessing values of $U(k)$ for $\forall k \in N_x$, the proposed PUP algorithm can adaptively promote some O_x's neighbor(s) with the highest utility value to S and also contributes to the rapid rescheduling of L_{dem} that is caused by the unexpected churn and layer loss. As discussed in Sect. 3.1, insufficient upload capacity or long forwarding latency can be also partially supplemented by relaying streams originated from other neighbors or Π in LSON.

A heterogeneity level of each peer in terms of connectivity can be determined by c_k in Eq. (10) because peers loosely connected enter and leave the network quite frequently, so that degree-dependent failures are commonly considered in this paper. Also, c_k can be influenced by a type of used graphs to model the super-peer overlay networks. In particular, we assume that LSON is constructed by perfect difference graphs, which are undirected inter-connection graphs with z vertices. Each peer has a degree of connectivity as $c_k = \mathrm{O}(\sqrt{z})$, so that LSON becomes more scalable than that modeled by other types of graph [9].

5 Simulation and discussion

5.1 Simulation setup

We extensively conduct simulations to evaluate the proposed layer scheduling scheme in LSON by implementing rmGA codes on the OverSim framework [22]. Each of 3,000 sub-streams is encoded into 10 layers according to the scalable video coding and layers are repeatedly transmitted until the end of the simulation. All layers have identical size (e.g., 1024 bytes) for ease of demonstration and the default streaming rate is 512 kbps.

In the simulation, we suppose that n is 1,000, m can be maintained within 5 % of n [8], and the number of connected super-peers given to each O_x is 15 at maximum. Since each O_x can averagely keep about 16 connections with its neighbors over time under assumption of using perfect difference graphs, we intentionally impose the limited number of super-peers to O_x in order to make O_x cooperate with its neighbors. Further, the inbound and outbound bandwidth of peers is assumed to be uniformly distributed.

Commonly used parameter configuration in the simulation is shown in Table 1.

Table 1 Parameter configuration

Parameters	Default value
Number of ordinary-peers	1,000
Maximum number of super-peers	5 % of n
Maximum number of connected super-peers for each O_x	15
A size of sub-streams, number of layers for a sub-stream	3,000, 10
Layer size	1024 bytes
Streaming rate	512 kbps
Population size	30
Maximum number of generations	250
Cut-and-Splices probability	0.18
Mutation probability	0.25

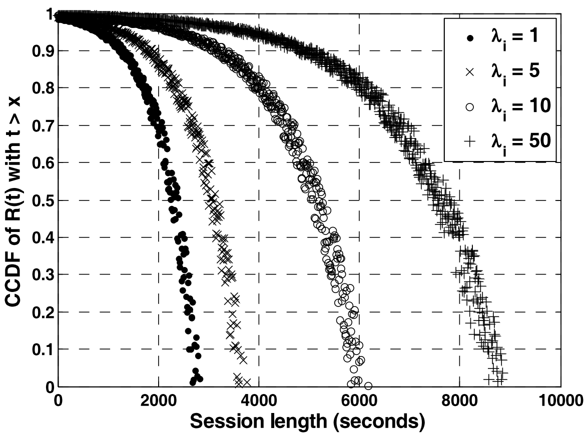

Fig. 4 CCDF of residual session lengths with $x = 1$ hour, $1 \leq \lambda_i \leq 50$

5.2 Behavior analysis of radom churn

In Fig. 4, when a lifetime of S_i, Φ_i, follows the Weibull distribution [20] as discussed in the study [19], we plot the percentage of remaining super-peers during the service time with different values of scale parameter λ_i, using the complementary CDF (CCDF) of $R_i(t)$ for $i = 1, 2, \ldots, m$. From the observation [2], we have $\Phi_i = \lambda_i \cdot (-\ln(\Omega))^{-1/s}$ as a generating function, where a shape parameter Ω is uniformly drawn from [0, 1] and s is approximately 0.4. $E[\Phi_i]$ is then obtained as $\lambda_i \cdot \Gamma(1 + 1/s)$, where Γ is a gamma function [20], so that the CCDF of $R_i(t)$ is finally given by $1 - H_i(t)$. As shown in Fig. 4, we can decide a degree of heavy-tailedness in $R_i(t)$ by adjusting a value of λ_i in $[1, \infty]$. When λ_i is closer to 1, S_i would be highly volatile, resulting in that the departure more frequently occurs. For instance, 41 % of super-peers with $t > 1$ hour are approximately remaining at 8,000 seconds, when λ_i is set to 50.

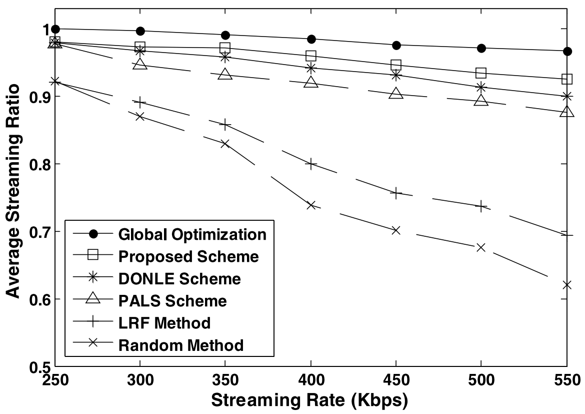

Fig. 5 Comparison of average streaming ratio under six different layer scheduling schemes ($10 \le \lambda_i \le 100$)

5.3 Average streaming ratio

Figure 5 compares the streaming quality of six different layer scheduling heuristics under the relatively stationary LSON, where the average streaming ratio is computed by dividing $\sum_{x \in X}(Q_x^T - Q_x^{\Pi})/r^+(t)$ by n. Over the whole range of streaming rate, the proposed scheme outperforms the other approaches by as much as 1.5–24.8 %; while, on average its result is 2.9 % smaller than the ideal solutions of global optimization. However, global optimization may not scale well with increasing size of LSON because it requires global knowledge.

5.4 Churn resilience assessment

To perform a comparative study of the proposed GALS algorithm using different promotion methods, we define three strategies targeting specific metrics such as $h_{k,x}$, $u_{k,x}$, and σ_k. The conventional shortest path first strategy (SPFS) and content availability oriented strategy (CAOS) use the hop count and the first two factors to assess an k's utility value, respectively; whereas, all metrics are combined into $U(\cdot)$ for the proposed PUP strategy (PUPS).

Figure 6 demonstrates the average bandwidth allocated to ordinary-peers versus different intensity of churn with the three strategies. More specifically, CAOS remains competitive to PUPS up to 80 in λ_i; however, a layer loss rate of CAOS is sharply increased compared to PUPS, as λ_i grows. On the average, PUPS outperforms others by 11.3 % and 28.9 %, respectively and guarantees the smallest variation in bandwidth.

Figure 7 depicts traces of unfairness in respect of load balance under the same conditions above. This metric is obtained as a standard deviation of average utilization. We observe that fairness deteriorates over the whole point of λ_i in all strategies; while, the average utilization is stably maintained as shown in Fig. 7. Consequently, the proposed GALS algorithm using PUPS offers the best availability and immunity to random churns in processing the requests of fetching layers under heterogeneous bandwidth limitations in LSON.

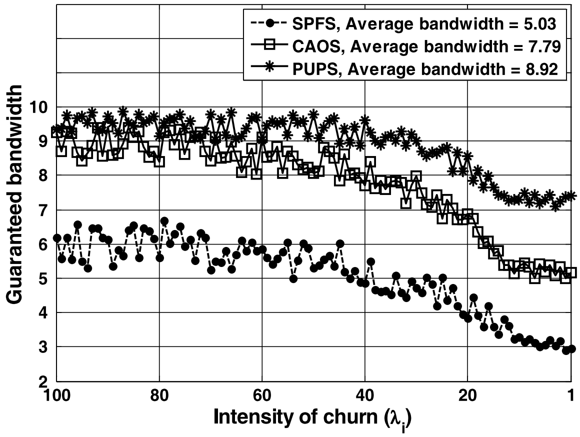

Fig. 6 Guaranteed bandwidth

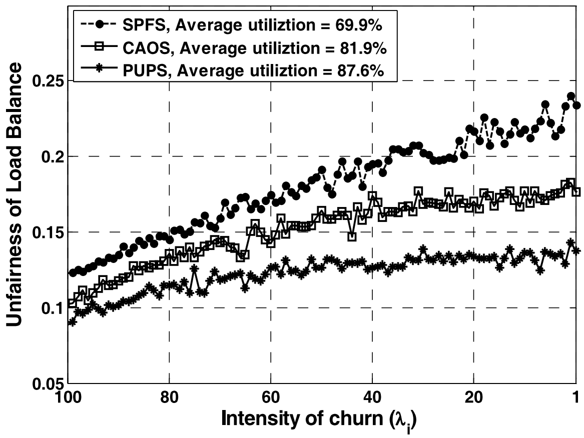

Fig. 7 Unfairness of load balance

5.5 Saving of server bandwidth

We assume that peer types are classified into four classes according to its inbound and outbound bandwidth capacity as shown in Table 2. For instance, peers having higher connectivity or uptime (e.g., connected through Ethernet) are more stable in the network than the peers having lower connectivity or uptime (e.g., connected through dial-up line).

In particular, the percentage of *Ethernet* peers, P_{EP}, is set to 5 % by default and those of others are randomly distributed. We investigate the effects of corresponding supplying peers of O_x on the server cost with different combinations of P_{EP} and neighbor selection methods such as PUPS and CAOS. For this simulation, we also use the GALS algorithm. Figure 8 shows that as the number of corresponding supplying peers is increased from 2 to 10, the consumption of average server bandwidth is reduced and PUPS is more effective in saving the server cost. In addition, high contribution of more-capable peers is desirable for LSON due to their abundant outbound bandwidth.

Table 2 Peer types

Peer types	Inbound/outbound
Mobile	784/128 kbps
xDSL	15×10^2/400 kbps
Cable	$3 \times 10^3/10^3$ kbps
Ethernet	$10^4/5 \times 10^3$ kbps

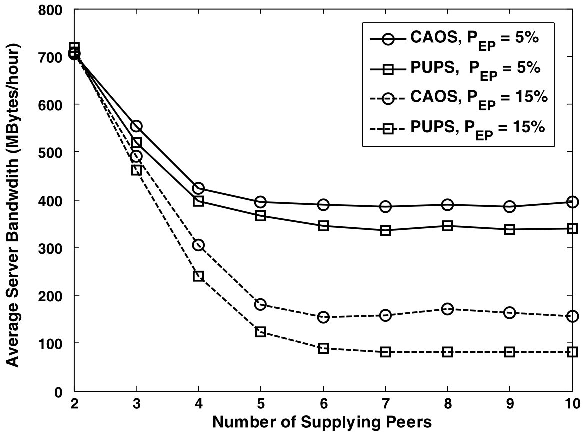

Fig. 8 Server bandwidth reduction

5.6 Simulation runtime

In this section, we seek to determine which $\acute{I}$ can make the layer scheduling problem computationally tractable under the given streaming conditions. Figure 9 plots the computing overhead of scheduling layered streams using the proposed scheme with 15 super-peers at maximum. For this simulation, since we assume that the request interval is 2.5 seconds, the proposed scheme can deal with 160 layers (i.e., 20 sub-streams) in 35–90 microseconds for every period. Namely, the best possible convergence time is guaranteed when $\acute{I}^{\text{opt}} = 160$ layers under the maximum of 15 super-peers and the average of 16 neighbors. This result is practically acceptable compared to the request interval.

6 Conclusion

In this paper, we have studied the problem of scheduling differently prioritized layers with the goal of maximizing the differentiated quality of streaming services and minimizing the adverse effects of churn on LSON, where incremental scaled stream processing is allowed over multiple upstream peers. Based on our understanding of the problem, main drawbacks have been investigated in respect of aggregated download bandwidth in the LSON model and then we have formulated a layer scheduling problem (in the form of optimal policy) for the differentiated layered streaming under the three types of constraint such as layer dependency, transmission rule, and bandwidth

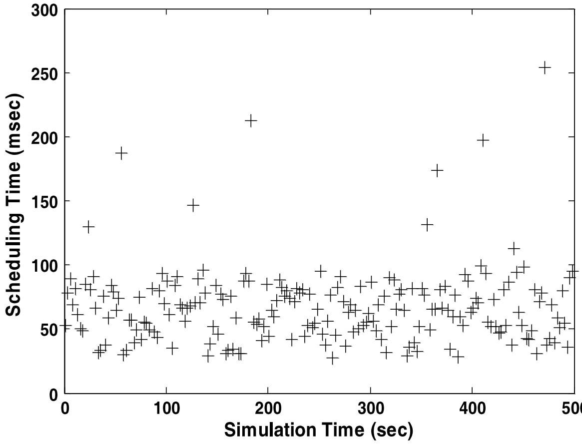

Fig. 9 Computing overhead of the proposed layer scheduling scheme. A maximum of 15 supplying peers and a request interval of 2.5 seconds are employed over LSON

heterogeneity. In order to solve this problem, we have proposed an rmGA based layer scheduling (GALS) algorithm, which can rapidly search for a near-optimal solution within the best possible convergence time. Further, we have proposed a peer-utility-based promotion (PUP) algorithm to improve decentralized manageability against the high rates of churn. Compared to the conventional scheduling methods and neighbor selection strategies, the proposed churn-aware optimal layer scheduling scheme has achieved high sustainability as well as low fluctuation as analyzed in various simulation results, while satisfying all the constraints discussed.

Acknowledgements This research was equally supported by R&D programs of MEST/NRF [2012-0020522, the Next-Generation Information Computing Development Program], and MKE/KEIT [10039260, Integrated development environment for personal, biz-customized open mobile cloud service and Collaboration tech for heterogeneous devices on server]. This research also was supported by IT R&D program of MKE/KEIT [10038768, The Development of Supercomputing System for the Genome Analysis].

References

1. Ibaraki T, Katoh N (1988) Resource allocation problems: algorithmic approaches. MIT Press, Cambridge
2. Stutzbach D, Rejaie R (2005) Understanding churn in peer-to-peer network. In: Proceedings of the ACM Internet measurement conference (ACM IMC)
3. Zhou X, Ge Y, Chen X, Jing Y, Sun W (2012) A distributed cache based reliable service execution and recovery approach in MANETs. J Converg 3(1):5–12
4. Pai V, Kumar K, Tamilmani K, Sambamurthy V, Mohr AE Mohr EE (2005) Chainsaw: eliminating trees from overlay multicast. In: Proceedings IEEE INFOCOM
5. Zhang X, Liut J, Lis B, Yum T-SP (2005) Coolstreaming/DONet: a data-driven overlay network for efficient live media streaming. In: Proceedings IEEE INFOCOM
6. Agarwal V, Rejaie R (2005) Adaptive multi-source streaming in heterogeneous peer-to-peer networks. In: Proceedings of the multimedia computing and networking (MMCN)
7. Zhang M, Chen C, Xiong Y, Zhang Q, Yang S (2007) Optimizing the throughput of data-driven based streaming in heterogeneous overlay network. In: Proceedings of ACM multimedia modeling (ACM MMM'07)
8. Xiao L, Zhuang Z, Liu Y (2005) Dynamic layer management in superpeer architectures. IEEE Trans Parallel Distrib Syst 16(1):1078–1091

9. Li J-S, Chao C-H (2010) An efficient superpeer overlay construction and broadcasting scheme based on perfect difference graph. IEEE Trans Parallel Distrib Syst 21(5):594–606
10. Kim H, Lee S, Lee J, Lee Y (2010) Reducing channel capacity for scalable video coding in a distributed network. ETRI J 32(6):863–870
11. Mastroianni C, Cozza P, Talia D, Kelley I, Taylor I (2009) A scalable super-peer approach for public scientific computation. Future Gener Comput Syst 25(3):213–223
12. Schwarz H, Marpe D, Wiegand T (2007) Overview of the scalable video coding extension of the H.264/AVC standard. IEEE Trans Circuits Syst Video Technol 17(9):1103–1120
13. Goldberg DE, Korb B, Deb K (1989) Messy genetic algorithms: motivation, analysis, and first results. Complex Syst 3:493–530
14. Wei Q, Qin T, Fujita S (2011) A two-level caching protocol for hierarchical peer-to-peer file sharing systems. J Converg 2(1):11–16
15. Luo H, Shyu M-L (2011) Quality of service provision in mobile multimedia—a survey. Hum-Cent Comput Inf Sci. doi:10.1186/2192-1962-1-5
16. Goldberg DE (1989) Genetic algorithms in search, optimization, and machine learning. Addison-Wesley, Reading
17. Moon Y-H, Youn C-H (2012) Integrated approach towards aggressive state-tracking migration for maximizing performance benefit in distributed computing. Clust Comput. doi:10.1007/s10586-011-0197-0
18. Lobo FG, Goldberg DE, Pelikan M (2000) Time complexity of genetic algorithms on exponentially scaled problems. In: Proceedings of the genetic and evolutionary computation conference, pp 151–158
19. Leonard D, Yao Z, Rai V, Loguinov D (2007) On lifetime-based node failure and stochastic resilience of decentralized peer-to-peer networks. IEEE/ACM Trans Netw 15(3):644–656
20. Ross SM (1996) Stochastic processes. Wiley, New York
21. Aikebaier A, Enokido T, Takizawa M (2011) Trustworthy group making algorithm in distributed systems. Hum-Cent Comput Inf Sci. doi:10.1186/2192-1962-1-6
22. Overlay and peer-to-peer network simulation (OverSim) framework. http://www.oversim.org/