Designing a Tit-for-Tat Based Peer-to-Peer Video-on-Demand System

Kévin Huguenin, Anne-Marie Kermarrec, Vivek Rai, Maarten van Steen

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Designing a Tit-for-Tat Based Peer-to-Peer Video-on-Demand System

Kévin Huguenin — Anne-Marie Kermarrec — Vivek Rai — Maarten van Steen

N° 7034

Septembre 2009

Domaine 3

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--7034--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
RENNES - BRETAGNE ATLANTIQUE

Designing a Tit-for-Tat Based Peer-to-Peer Video-on-Demand System

Kévin Huguenin , Anne-Marie Kermarrec , Vivek Rai*, Maarten van Steen*

Domaine : Réseaux, systèmes et services, calcul distribué
Équipe-Projet ASAP

Rapport de recherche n° 7034 — Septembre 2009 — 20 pages

Abstract: Video-on-demand (VoD) is a next-generation Internet application of increasing interest allowing users to start watching a movie almost instantaneously by downloading the video on-the-fly. Provided that all users contribute to the system, shifting to the P2P paradigm allows efficient broadcast with a limited-bandwidth source. Until now, most P2P VoD systems rely on the willingness of peers to collaborate and only few use incentive mechanisms. In VoD applications pieces are downloaded in order. This prevents to directly apply a BitTorrent-like tit-for-tat incentive scheme. Although existing solutions based on random exchanges have good performance, we advocate the use of structure in P2P VoD applications to achieve high playback rates. In this paper we propose a decentralized piece dissemination scheme built using loosely coupled structures. Peers are grouped into clusters depending on their playback position. Swarming is performed inside the clusters while distributed feeding ensures that less advanced clusters get missing pieces from more advanced ones. Our simulations demonstrate that structured dissemination improves from 61% with the competitor to 77% of the achievable playback rate.

Key-words: Peer-to-peer systems, Video on Demand, Tit-for-Tat

* Vrije Universiteit, Amsterdam, The Netherlands

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex
Téléphone : +33 2 99 84 71 00 — Télécopie : +33 2 99 84 71 71

Conception d'un système de vidéo à demande pair à pair basé sur l'incitation

Résumé :

Mots-clés : Réseaux pair à pair, vidéo à la demande, méchanismes incitatifs

1 Introduction

Video-on-demand (VoD) is a next-generation Internet application of increasing interest allowing a user to start watching a movie of his choice almost instantaneously. The media content is downloaded during the playback in order to ensure that every piece of the media is available at the device when the playback position reaches it. Therefore, at the price of a small time delay (i.e., compared to the naive solution consisting in downloading the full movie before starting the playback), the movie can be played smoothly without interruption. The high bitrates of the broadcasted content and the quality of service requirements of VoD applications make centralized solutions costly. Due to the inherent sequential nature of VoD transfers, building efficient decentralized VoD applications is more challenging than traditional client-server solutions.

Over the last decade, the peer-to-peer (P2P) paradigm proved to be an efficient way to distribute content in a decentralized fashion [4, 20]. A very popular P2P swarming protocol enabling file sharing between peers is BitTorrent [7]. With only a limited number of peers injecting content in the system (namely *seeders*) and proper forwarding techniques performed at the peers fetching content from the system (namely *leechers*), P2P systems provide a fully decentralized distributed framework allowing efficient content distribution at low cost. Therefore, they appear as a natural cheap solution for VoD applications. However, applying the P2P paradigm to VoD systems is a difficult problem for two reasons: *(i)* constraints on the piece download order and the low piece diversity in the system, decrease drastically the performance of the system and *(ii)* considering the fact that the download speed relies on possibly selfish peers it is impossible to guarantee any kind of quality of service to the users.

While the first problem can be fixed by designing proper piece dissemination schemes, the latter requires a total rethinking of the P2P paradigm: incentives should be used to encourage peers to contribute their fair share to the system. The tit-for-tat distributed incentives used in the popular file sharing system BitTorrent have proved to ensure strictly that a peer contributes to the swarming process as much as it exploits the system [8]. Tit-for-tat is implemented by limiting piece exchange to a bidirectional transfer between peers having mutual interest. Although tit-for-tat solves the problem of selfish peers, using them in a VoD application – where the peers only have the pieces before their playback position – raises the question of how two peers at different playback positions could be useful for each other?

Motivated by the increasing interest of VoD and inspired by the most popular file-swarming application deployed in the public domain (i.e., BitTorrent), we tackle the problem of designing a fully decentralized tit-for-tat based peer-to-peer VoD system. More specifically, backed up by theoretical arguments we prove that whereas the efficiency of random solutions is close to optimal in file-sharing applications with offline use, it is strongly reduced by the specific nature of VoD applications. Even if a random piece dissemination scheme can achieve good performance by means of several optimizations [3, 16, 18], we claim that the inherent sequential nature of VoD is not compatible with tit-for-tat using a random piece exchange scheduling and communication graph.

In this paper we propose the design of a fully decentralized protocol for efficient tit-for-tat based peer-to-peer VoD systems. This protocol trades the traditional random piece dissemination scheme against a loosely structured dissemination scheme based on linked lists following the intuition below. Ordering peers according to their playback position and linking them that way achieves an optimal throughput as all peers contribute to the piece dissemination. This ensures a constant goodput over time, a useful piece being disseminated at each exchange. The resulting protocol achieves this in a practical setting by grouping peers in subsystems. This solution relies on two main components. First, a set of swarming subsystems, referred to as clusters, enabling peers close with respect to playback time, to exchange pieces of immediate interest as in traditional swarming systems. In the sequel, we call this vertical dissemination. The second component is a distributed seeding/feeding protocol used to exchange pieces between clusters. The intuition is that the most advanced clusters, in terms of playback position, feed the less advanced ones, while the less advanced clusters push useful pieces to the most advanced ones. Those useful pieces, from the standpoint of the most advanced clusters, are provided by the seed to the less advanced clusters, in order to provide them with some useful piece to trade. This is called horizontal dissemination as it builds a linked list between clusters along the playback timeline.

We evaluated this protocol through extensive simulations on top of the BitTorrent framework. This protocol consistently outperforms the state of the art VoD system. Typically this protocol improves the achievable playback rate from 61% to 77% and the throughput from 68% to 87%. Moreover, our simulations show that, contrary to an unstructured piece dissemination protocol, our dissemination

protocol does not require the peers to store all the pieces they already played. This implies that the protocol will also work with the ressource-constrained devices such as set-top boxes [12, 22]. Beyond a given threshold (independent from the file size and very small compared to the number of pieces in the file), our protocol achieves a playback rate of 77% whereas the unstructured protocol may fall to 18%.

Section 2 provides the relevant work in the design of peer-to-peer VoD systems. Section 3 gives an overview of the BitTorrent framework on top of which we designed our algorithm. Section 4 gives our design rationale. Section 5 provides a high-level description of our algorithm, followed by a detailed description in Section 6. There, we also show how to provide a fully decentralized implementation. Protocol analysis and simulation results are given in Section 7 and Section 8. The paper is concluded in Section 9.

2 Related Work

Most of the large-scale peer-to-peer content dissemination schemes are designed using a tree-like structure. A tree provides a natural topology when content is pushed from a single source to multiple destinations. However, an important drawback of a tree-based dissemination scheme is that the upload bandwidth available at the leaf nodes is wasted. Therefore, solutions such as using multiple trees are proposed in order to maximally utilize the bandwidth resources available at the peers. SplitStream [4] solves this problem by striping the content across a forest of inter-node-disjoint trees. Other solutions have also been proposed including [6, 9]. A general argument against the above discussed multi-tree based approach is their relatively higher cost of maintenance in a dynamic environment [15]. In addition, since the transfer of content is not bidirectional, they are not compatible with tit-for-tat based incentive models.

Several mesh based solutions such as BitTorrent [7] have also been proposed for content dissemination [14]. The advantage of such a solution is high scalability due to decentralization. Furthermore, BitTorrent incorporates a tit-for-tat based incentive scheme to prevent free-riding and to encourage peer contribution to the dissemination process. In BitTorrent-styled file dissemination, a file is divided into multiple pieces such that each piece is independently downloaded. The incentive mechanism implies that in order to download a piece, a peer must upload a piece in return.

Diversity of pieces available at the peers is critical for the success of these mesh-based P2P file dissemination technique. However, in a VoD application, the restriction on the order in which pieces are downloaded limits the piece diversity. Furthermore, allowing tit-for-tat based incentives may be difficult to achieve as two peers at different playback position cannot be of mutual interest.

Several solutions have been proposed with focus on improving piece diversity in order to increase the performance of the system. A trivial solution introduced in [17] consists in prefetching some pieces randomly while the rest is downloaded in a sequential order. The motivation for downloading pieces in a random order is to achieve high piece diversity and provide less advanced peers with pieces to trade with more advanced ones. A piece downloaded in a random order can be exchanged with a more advanced peer and in return a piece near the current playback position can be obtained. Random downloads help in increasing the throughput of the swarming process, whereas downloading pieces in a sequential order is critical for the performance of the VoD application. Finding a critical balance between random and sequential downloads is an important design challenge.

Increase in piece diversity can also be achieved by slightly relaxing the sequentiality requirement by using the segment model introduced in [2]. In this model, a file is divided into segments, where a segment is a group of continuous pieces. Rather than downloading all the pieces in a sequential order, peers download the file at the segment granularity. To guarantee uninterrupted playback, a peer has to ensure that each segment is completely downloaded before the playback position reaches the beginning of that segment. Note that, even though the segments are downloaded in a sequential order, there is no restriction on the order in which pieces are downloaded within a segment. This provides a high piece diversity within a segment. However, as segments are downloaded in sequential order, this results in only a small increase in the piece diversity over the entire swarm.

Notice that none of these solutions are designed to work with decentralized incentives such as tit-for-tat. These may work with tit-for-tat, however, with reduced performance.

3 Background

Peers participating in a BitTorrent swarm often are profit maximizing entities and their objective is to optimize their download rate. BitTorrent provides an incentive mechanism that allows peers to independently decide which other peers they want to exchange their content with. The incentive mechanism ensures that in order to successfully exchange content, a peer must provide upload bandwidth comparable to the download rate it receives.

The piece transfers in tit-for-tat based file-swarming systems are done over bidirectional connections. Due to this bi-directionality restriction on piece exchange, the number of pieces that can be exchanged over the entire swarm is reduced. Thus, to maximize throughput of the swarm, peers maintain piece diversity by downloading different parts of the file such that every peer has a high probability of exchanging content with another.

Sufficient piece diversity can be obtained if peers download pieces in a random order. The random order piece download serves the objective of the file-transfer application where the entire file is downloaded before it is viewed. However, for a VoD application, there is a restriction on the order in which pieces are downloaded. For example, for a simple VoD application, where every peer plays the file from start to finish, it is very difficult to maintain sufficient piece diversity to attain high throughput of piece exchange. Thus, this lack of piece diversity due to sequentiality requirements of the VoD application can severely limit the throughput of the system and hence can significantly reduce the performance of P2P file-dissemination techniques.

To further maximize its chances of exchanging pieces, each peer maintains a list of other peers with whom it can exchange content. This list of peers is also known as the peer set. A peer keeps an update on the set of pieces downloaded by the members of its peer set. Using this piece set information, a peer can locally determine potential candidates within its peer set with whom it can exchange pieces. Therefore, several unnecessary requests to the peers that are not interested in piece exchange can be saved. The choice of the members within a peer set is very crucial, especially for the design of VoD application. In BitTorrent, members of the peer set are chosen randomly.

4 Design Rationale

In this section, we explore the design space of tit-for-tat based P2P VoD.

4.1 Design rationale

Consider a simple example of a VoD system with two participants denoted by n_1 and n_2. We assume that n_2 is at a more advanced position compared to n_1 such that the piece set of n_2 forms a superset of that at n_1. Thus, even though n_2 can upload the piece needed by n_1 it cannot receive anything in return as n_1 does not have the piece needed by n_2, and hence piece exchange is not possible. In order to make possible an exchange between two peers, a seed can upload the pieces needed by n_2 to n_1 such that n_1 can further exchange those pieces with n_2. In addition, this enables the bandwidth of both peers to be utilized for the dissemination process. Extending this idea for more than two participants, we now consider k nodes such that nodes n_k to n_1 are arranged in decreasing order of their playback position as depicted in Figure 1. Therefore, the piece set of an intermediate node n_i ($1 \leq i \leq k$) is a superset of the piece sets of all the nodes with lower playback position. Now, an optimal throughput can be obtained if the seed uploads the piece required by n_k to n_1, and n_1 forwards this piece via all the intermediate nodes on a forward path that eventually reaches n_k. Similarly a reverse path can be obtained where node n_i downloads pieces in sequential order from n_{i+1} in return for pieces transfered on the forward path.

From the above discussion we can conclude that a natural structure to support efficient utilization of upload bandwidth is to arrange peers in a linked list. This linked list structure implies that half of the upload bandwidth is utilized on the forward path for forwarding pieces from the seed to the most advanced peer while the other half is dedicated to the reverse path for uploading pieces from a more advanced peer to a less advanced peer in sequential order. Consider the example depicted in Figure 2. There are five peers in the swarm with playback positions at 5, 13, 13, 22, and 34. Therefore, the most advanced piece missing from the swarm is 35. The seed pushes piece number 35 to the least advanced peer, i.e., the one at playback position 5. Piece number 35 is further uploaded on the forward path to

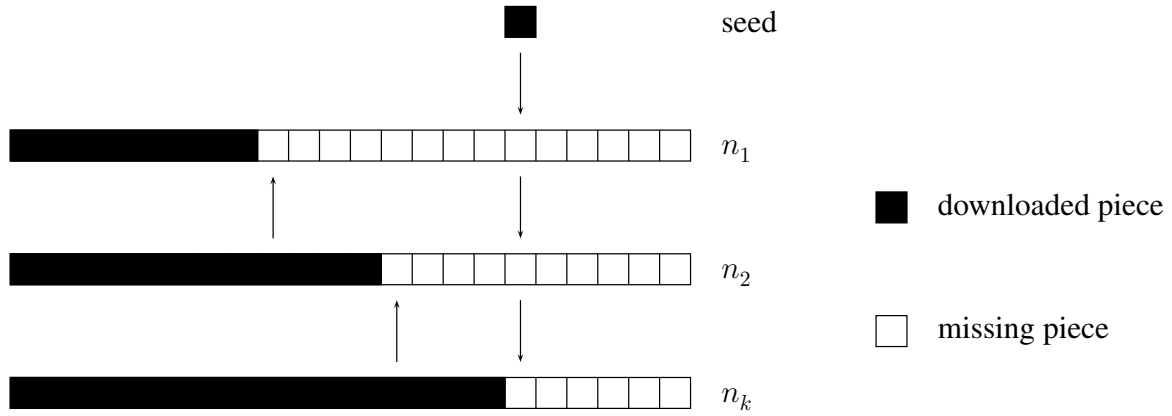


Figure 1: The seed forwards the most advanced piece to the least advanced peer. This piece is eventually uploaded to the most advanced peer on a forward path. In return, in order pieces are downloaded on a reverse path.

eventually reach the most advanced peer. In exchange for piece number 35, each peer downloads a piece in sequential order. Note that the pieces transferred on the forward path are not for immediate use by any intermediate peer, but they still contribute to the overall throughput of the swarm. However, the pieces downloaded on the reverse path are in sequential order (i.e., they are the ones needed next by the receiving peer) and therefore contribute to the goodput (i.e., the proportion of downloaded pieces that are of immediate interest). Hence, a linked list structure can achieve theoretically optimal 100% throughput. However, as only half of the pieces are transfered in the desired order, which implies that the overall maximum achievable goodput is 50%. Downloading one out of every two pieces in the right (sequential) order while attaining very high throughput is a significant improvement compared to file swarming where there is no guarantee on the order in which pieces are downloaded. However, it is yet to be established whether the linked list structure can be maintained in a practical setting.

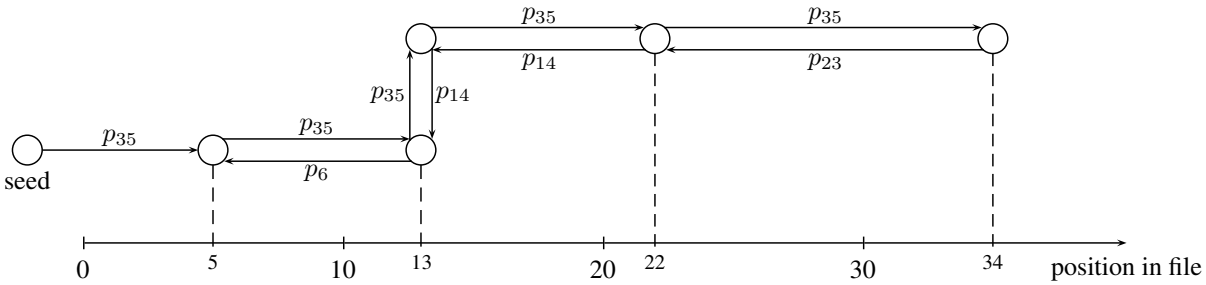


Figure 2: Piece exchange process under a fluid model

4.2 Practical considerations

In a practical file-swarming system such as BitTorrent, a piece can be forwarded only after it has been completely and successfully received. There are several challenges that need to be considered. Consider the example in Figure 2, where there are two peers at position 13. Under a full piece exchange requirement, two peers can exchange a piece only if both of them have at least one piece different from the other. Therefore, two peers at the same position cannot exchange a piece until both of them are simultaneously fed with two different pieces. Hence the linked list structure cannot be maintained without removing one of these two peers from the list as depicted in Figure 3. Thus, under the full piece exchange requirement, if there are multiple peers at the same playback position they cannot remain in the same linked list.

This results in the linked list structure getting separated into multiple lists, which implies that the seeding capacity should now be divided to feed multiple lists and hence it results in an inefficient

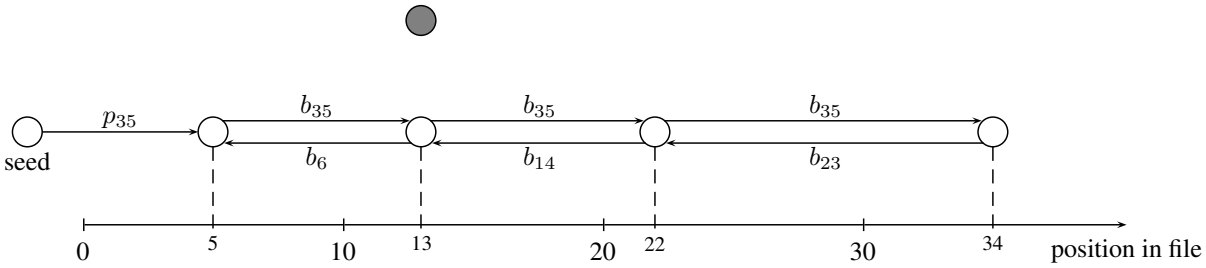


Figure 3: Piece exchange process under a full piece upload model

utilization of the seeding capacity. In addition, due to the full piece exchange requirement, the delay incurred by a piece to reach the most advanced peer is proportional to the length of the list. The tit-for-tat incentive model used in BitTorrent implies that piece exchange has to be simultaneous. Therefore, the contract between two peers is only guaranteed for one piece exchange. However, in order to ensure the delivery of the most advanced piece on the forward path, we have to guarantee that the contracts on all the intermediate peers are maintained for the duration proportional to the length of the list. Ensuring long duration contracts between multiple peers in a decentralized setting is a difficult problem. Therefore, the linked list constructed in a practical setting is not a viable solution. Consider the example depicted in Figure 4(a) with multiple peers at position 34. All these peers at position 34 are split into different linked lists. Note that the peer at position 22 contains piece 35 and that it can forward that piece to only one of the peers. Thus, the peer at position 22 is a bottleneck. Similarly consider the example depicted in Figure 4(b), where there are multiple peers at position 22. However, only one of them contains piece 35. Therefore, it will take several rounds after which piece 35 can be forwarded to the peer at position 34. The above problems appear only due to the lack of piece exchange between the peers at the same position. This justifies the need for vertical piece exchanges.

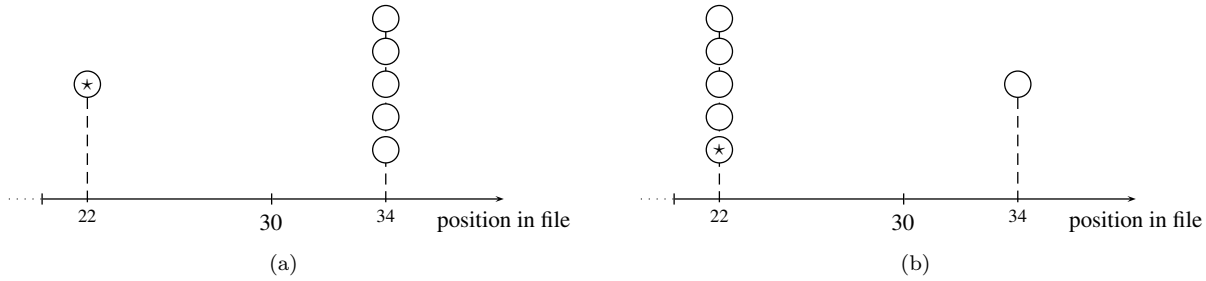


Figure 4: Independent linked list

5 Structured Piece Dissemination

In the previous section, we identified that a linked list structure is a natural solution for providing tit-for-tat based VoD, which can achieve maximum throughput. However, due to several practical reasons such as full download requirement, multiple peers at the same playback position, etc., we conclude that a solution based on a single peer-level linked list is not viable. A natural extension is to maintain several linked lists, which are seeded separately as illustrated in Figure 5. The seeding capacity can be equally divided between all the linked lists. However, this solution is also not scalable because the number of linked list grows with the size of the system, and it also suffers from poor performance due to lack of piece exchange between peers at the same playback position.

Remember that the solution proposed in [2] relaxes the sequentiality requirement by splitting the file into segments, which are downloaded sequentially. However, the pieces within a segment are downloaded in a random order. The set of peers downloading the same segment can now be considered as an independent swarm (referred to as a cluster) such that each of these clusters can be seeded separately. In

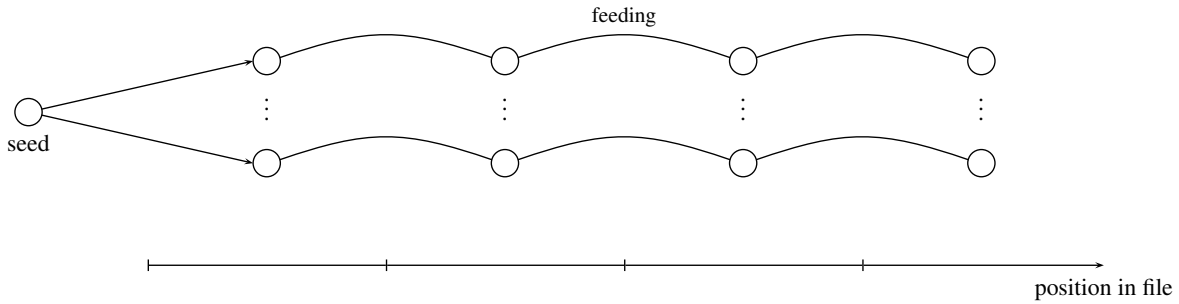


Figure 5: Distributed feeding using multiple linked lists at the peer level (horizontal piece exchanges)

this way, only the vertical exchanges are performed such that pieces are exchanged between peers within the same segment. If sufficient seeding capacity could be provided to all these clusters then a very high throughput can be attained. A simple way to provide seeding is to equally divide the seeding capacity between clusters as demonstrated in Figure 6. However, if the number of clusters is very high, then the available seeding capacity may not be sufficient.

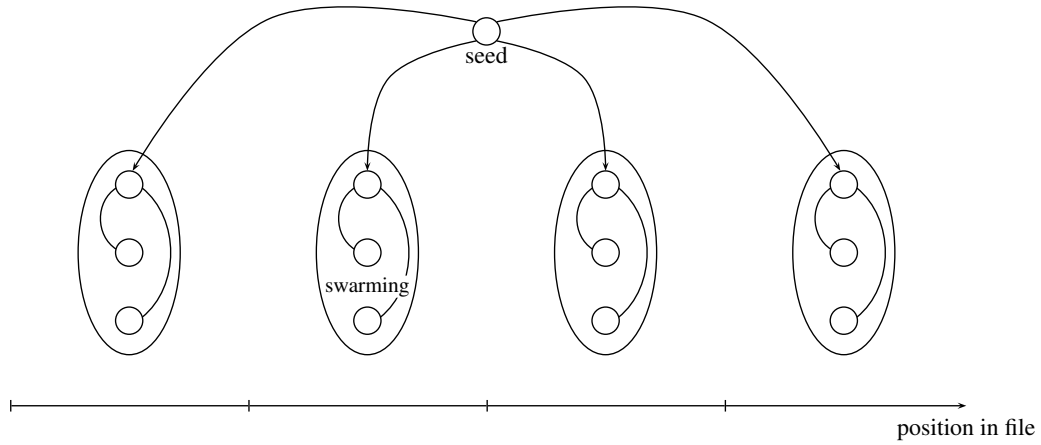


Figure 6: Independent swarming with centralized seeding (vertical piece exchanges)

Notice that the distributed feeding technique as depicted in Figure 5, which is based on horizontal piece exchanges is complementary to the independent swarming technique shown in Figure 6 that is based on vertical transfers. However, neither of these two techniques is sufficient enough by itself. Therefore, we design a *hybrid* scheme where the *vertical piece transfers are utilized for swarming* and *horizontal piece transfers are used for distributed feeding* (see Figure 7). In order to achieve that, we allow the peers within the same segment to be grouped together into clusters. These clusters are then fed with pieces in a distributed manner using a linked list structure. This hybrid scheme solves all those problems that we encountered in a similar linked list styled piece dissemination structure at the peer level.

There are several advantages to this hybrid scheme, for example, the length of the list is now limited by the number of segments which is a constant and does not vary with the size of the swarm meaning that the delay remains constant. Further, facilitating piece transfers within a segment allows for vertical transfers between peers in the same segment and hence eliminate the bottlenecks described before. Note that even in this model, the horizontal piece exchanges are done over a linked list. However, these linked lists are not independent such that they can be fed through each other. In a linked list at the peer level, every piece has to feed at least one piece on its forward path in order to download one piece on a return path in a sequential order. Therefore, a linked list at the peer level can achieve a maximum goodput of only 50%. However, in the cluster model, all peers within a cluster are responsible for feeding pieces. Therefore, the fraction of bandwidth utilized per peer in feeding pieces is significantly reduced and hence a *much higher* goodput can be expected.

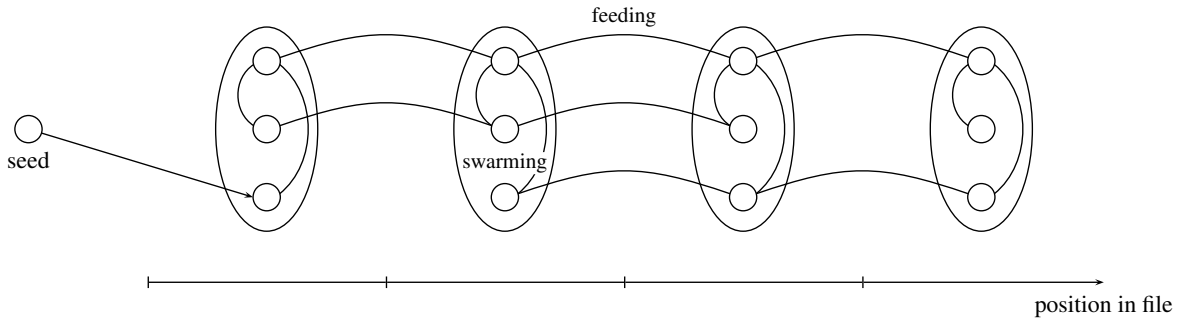


Figure 7: Hybrid solution at a cluster level

6 Algorithm in details

In this section we develop algorithms to implement structured piece dissemination, as described in the previous section, using the BitTorrent framework. Structured piece dissemination is essentially distributed feeding using multiple linked lists at the cluster-level together with swarming within a cluster over a random graph. As we mentioned earlier, in order to make distributed feeding more effective, the forward path should be as long as possible such that more clusters can be fed along the reverse path. To achieve these goals, we design our algorithm to *(i)* facilitate piece exchanges between peers within the same cluster, *(ii)* maximize the distance between the source cluster (to which the advanced pieces are seeded) and the target cluster (for which these pieces are of immediate interest) in terms of the playback positions; and *(iii)* maximize the number of intermediary clusters participating to the forward/reverse path.

To this end, we need to make three important modifications to the traditionally used algorithms/policies used in BitTorrent. First, we introduce an alternate seeding policy. In BitTorrent, the objective of the seed is to provide rare pieces to the system. However, our objective here is to enable piece exchanges between peers in different clusters thus establishing as long as possible bi-directional linked list structures for piece exchange, i.e., distributed feeding. Second, we introduce peer set management to allow efficient swarming inside clusters and maximize the number of intermediary clusters involved in linked lists. This should be performed dynamically since the peer set must evolve with the download progress. Finally, we change the piece exchange policy. In BitTorrent, peers exchange a locally rare piece in order to maintain high piece diversity in the swarm. However, in VoD the objective is to establish a balance between swarming and feeding, therefore, an alternate piece exchange policy is needed.

Seeding Policy An important component required for the construction of a linked list at the cluster level is to identify the pieces required by the most advanced cluster. Since the pieces required by the most advanced cluster are not available in the swarm those pieces are provided by the seed. However, instead of directly uploading to the most advanced cluster, the seed provides those pieces to the least advanced cluster. This allows the least advanced cluster to have a good bargaining power in the system. The exchange policy has to be designed accordingly so that these pieces are further fed on the forward path such that every intermediate cluster downloads it and forwards it to the next cluster and eventually these pieces reach the most advanced cluster. Note that the seed can easily obtain a list of peers in the least advanced segment from the tracker. The most advanced segment can be obtained by the seed in a distributed fashion by polling peers through the linked list. Therefore, the seeding policy has negligible overhead.

Peer Set Management Remember that at any given time each peer is a participant of exactly one cluster based on its current playback position. The constituents of each cluster can exchange pieces among themselves and they can participate in a linked list style feeding process. Therefore, we ensure that the peer set of every node is limited to the peers either from the same cluster or from the neighboring clusters as depicted in Figure 8. When a new peer joins the swarm it connects with some peers within the first group. The remainder of the peer set is constructed by connecting with neighbor of neighbors.

Algorithm 1 Seeding policy

Input Seeding
  s: seeding capacity
  S^+: most advanced segment
  S^-: least advanced segment
  for i **from** 1 **to** s
    $n \leftarrow$ random peer in S^-
    $p \leftarrow$ random piece in S^+
    push p **to** n
  end for

During the course of download, it is important to maintain the linked list structure. The peer set is updated periodically. If the peer remains within the same cluster it asks its neighbors to return a subset of the peers from their respective clusters. When a peer moves out of a cluster, it should update its neighborhood such that it is now connected to the peers within its new cluster and also to some peers in the clusters neighboring to this new cluster. This can again be done by polling through neighbor of neighbors. This way we can easily maintain the structure in a decentralized way.

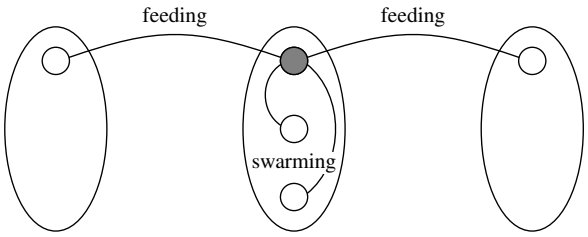


Figure 8: Peer set of a peer

Exchange policy The exchange policy determines whether two peers n_1 and n_2 should exchange pieces or not upon an encounter and which specific pieces p_1 and p_2 should be exchanged if any. If the peers are in the same group (i.e., their positions in the file lie in the same segment $S_1 = S_2$) then traditional swarming should be performed. Both peers look in a random order for a piece in their common current segment $S = S_1 = S_2$ that they could upload to each other. More specifically, they look for a piece in their piece sets that does not belong to the other peer's piece set. To ensure piece diversity inside each segment, and thus efficient intra-group swarming, such pieces are looked for by exploring the segment in a random order. Due to the peer set structure described in the previous paragraph advanced pieces can only be pushed from a cluster to the immediate next one when a peer connects to a member of the next cluster. In that situation the less advanced peer – say n_1 – downloads a randomly chosen useful piece for its current segment in exchange for a piece in the future. Priority is given to the most advanced pieces in segments after n_2's segment (denoted $p_2 > S_2$). If no such piece can be exchanged, then n_1 tries to upload a random piece in S_2. As explained in the previous sections, the motivation for uploading most advanced pieces with highest priority is two-fold: *(i)* ensure fast feeding of the most advanced segment and *(ii)* build an as long as possible forward path and thus a long reverse path which actually establishes intercluster feeding. If no mutual interesting pieces can be found using this exchange policy, the contract between the two nodes is simply broken. A pseudo-code version of the piece selection algorithm is given by Algorithm 2.

Note that using the transfer strategy presented in the previous paragraph, a peer uploads pieces in a previous segment only to peers in the previous groups whose playback positions lie in that given segment. In addition, due to the peer set management policy, a peer exchanges pieces only with the peers in the neighboring clusters (i.e., the cluster it belongs to, the previous and the next ones). Therefore, a peer can drop pieces before the playback position of the peers in the previous group without reducing its feeding ability. This specificity highlights the fact that structured protocols optimize resource utilization in terms of storage and bandwidth by making *only* the most appropriate peers uploading (and thus storing) *only* pieces needed by the peers they are in charge of. Therefore both storage and bandwidth load required by the application are distributed and balanced amongst the clusters: the most advanced

Algorithm 2 Piece exchange policy

Input Upon encounter of peers n_1 and n_2 (*assume* $n_1 \leq n_2$)
P_1 (resp P_2): piece set of n_1 (resp. n_2)
S_1: n_1's current segment
if n_1 and n_2 are in the same segment $S = S_1 = S_2$ **then**
  if $\exists p_1, p_2 \in S$ such that $p_1 \in (P_1 \cap \overline{P}_2)$ and $p_2 \in (P_2 \cap \overline{P}_1)$ (p_1, p_2: *random order search*) **then**
    exchange p_1, p_2
  else
    no exchange
  end if
else $\{n_1 < n_2\}$
  if $\exists p_1 \in S_1, p_2 > S_2$ such that $p_1 \in (P_1 \cap \overline{P}_2)$ and $p_2 \in (P_2 \cap \overline{P}_1)$ (p_1: *random order search*, p_2: *decreasing order starting from the end of the media*) **then**
    exchange p_1, p_2
  else if $\exists p_1 \in S_1, p_2 \in S_2$ such that $p_1 \in (P_1 \cap \overline{P}_2)$ and $p_2 \in (P_2 \cap \overline{P}_1)$ (p_1, p_2: *random order search*) **then**
    exchange p_1, p_2
  else
    no exchange
  end if
end if

peers do not use more bandwidth or more storage. The limited memory requirement of a structured piece dissemination (i.e., a small and constant number of segments) extends the application field of P2P VoD applications to the increasingly popular set-top box which operates with limited capabilities.

7 Discussion

In this section, we analyze several traditional protocol design issues in peer-to-peer.

7.1 Protocol Stability

First we consider the stability of the protocol. By protocol stability we mean whether the system will continue to function at its optimal level over time. The protocol specifies that the seed pushes pieces from the most advanced segment to the peers in the least advanced segment. When peers arrive at a regular rate, the last segment is most likely to be the most advanced segment.

One might argue that since the pieces are forwarded using intermediate clusters, the pieces of the last segment become highly replicated in the system gradually loosing their bargaining power. This may eventually lead to a situation in which the system gets stuck, since the peers in the least advanced clusters cannot get useful pieces in exchange of pieces in the last segment. In fact this never happens since the seed will automatically start pushing pieces from the next-to-last segment. Indeed, the pieces from the last segment are downloaded by the peers before they reach the last segment. Thus the last segment is no longer the most advanced segment as peers leave the system before they reach the last segment. The next-to-last segment will therefore become the most advanced segment until all those peers which have downloaded the last segment have departed from the system. When that happens the most advanced segment shifts back to the last segment. Thus, the most advanced segment will oscillate between the last and next-to-last segment. Hence the protocol is self-stabilizing.

Figure 9 plots the index of the most advanced segment (the file has been divided into 10 fixed-size segments) as a function of time, over a sample run of 1000 rounds. Peers join the system at a fixed rate (Poisson process of intensity 5 peers/round). During the transient state the most advanced segment progressively moves to the last one (i.e., index 9). Then, in steady state, the index of the most advanced segment oscillates between 9, 8 and 7 as foreseen in the previous paragraph.

7.2 Bottlenecks

We previously noted that there may be several bottleneck scenarios in the peer-level linked list. We now determine that bottlenecks can be easily prevented for the distributed feeding process in the cluster-level

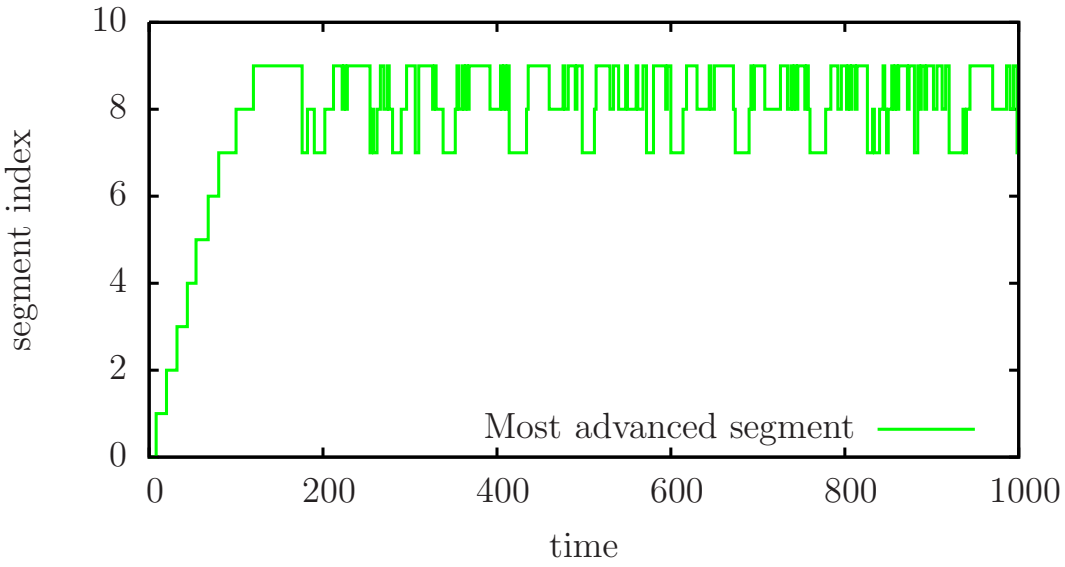


Figure 9: Evolution of the most advanced segment.

linked list. A bottleneck can occur if the size of a cluster falls below a certain critical value such that the cluster is not able to feed the next cluster, which subsequently results in all the subsequent clusters getting starved. Therefore, if a cluster is able to feed an entire segment in one round, then that will ensure that there is no starvation. To illustrate this point, consider the case where a segment size is 20 and the fraction of bandwidth employed for the feeding process is 50%. The bandwidth required to download a segment is 20 pieces per round. Assume that on average a peer can upload 4 pieces per round. Then, there should be at least 10 peers in the cluster in order to prevent the occurrence of a bottleneck.

Notice that the bottlenecks may not be an issue in large-scale dissemination, that is, where the cluster size is sufficiently big. On the other hand, with limited-scale dissemination, we may be able to determine or accurately estimate the system size in advance. In that case, the number of clusters can be controlled such that it is highly unlikely that the size of the cluster drops below the critical value. Otherwise, if the system size is not known in advance, we can dynamically adjust the size of the clusters by merging two consecutive clusters, whenever the size of one of the two clusters drops below the critical value.

Figure 10 represents the minimum, average and maximum sizes of the groups (the swarm has been divided into 10 groups, matching the 10 segments, according to the playback position of the peers) in steady state (i.e., after 500 rounds) over sample runs. All the intermediate groups (i.e., 2-6) are of similar sizes (i.e., 45 on average). The first group is smaller because peers join with one piece in that segment. Therefore they spend less time in that group and its size is thus reduced (according to Little's law [11]). The last groups have much smaller size because the pieces from the corresponding segment are downloaded by the peers before their playback positions reach these segments (due to the distributed feeding process through the link list). One can observe that the size of the first seven groups (responsible for the feeding process, the last three being in turn the most advanced segments) never drops bellow 20 peers which is sufficient for the feeding process on the direct path. This demonstrates that moving from a linked list at the peer level to a loosely coupled structure at the group level prevents bottlenecks in the feeding process.

7.3 Heterogeneity

In order to ensure tit-for-tat in an heterogeneous environment, the system may compromise on the performance due to the limited peer-set size. For example, if the upload bandwidth of a certain peer is greater than the total bandwidth available at all the peers within its peer set, the available capacity at the faster peer may not be fully utilized. To address this issue, the size of the peer set should be adjusted taking into consideration the heterogeneity in the upload bandwidth of the participating peers.

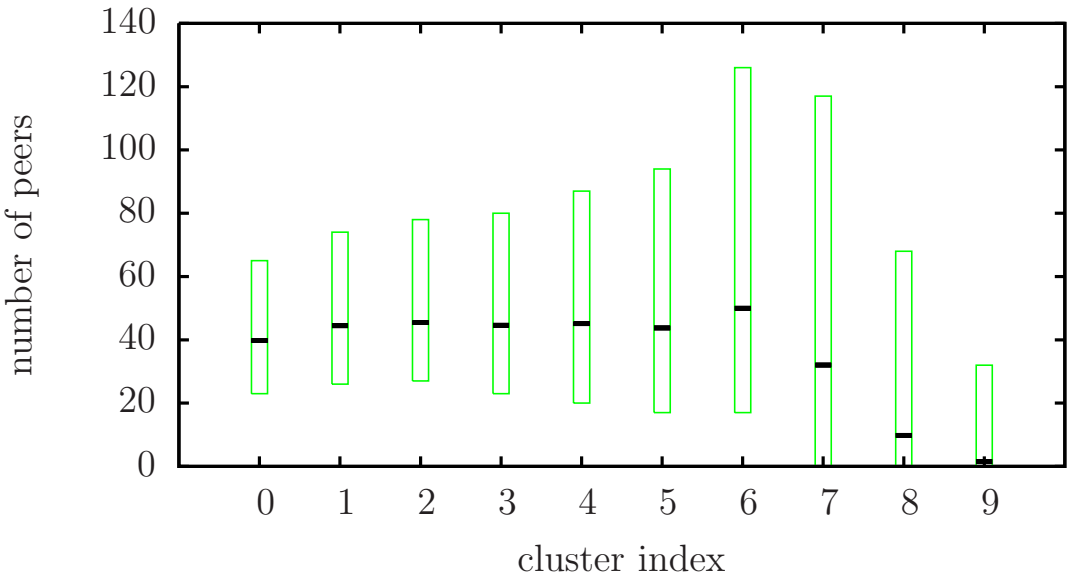
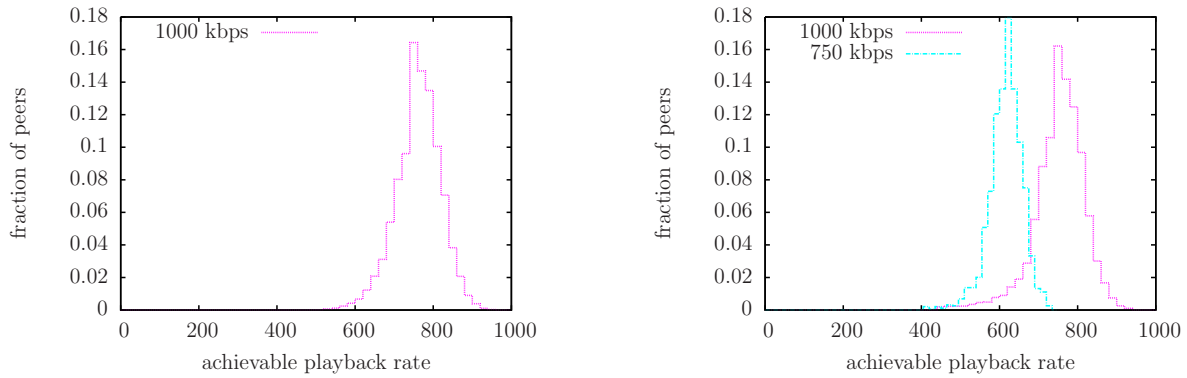


Figure 10: Distribution of the cluster size.

More specifically, the size of the peer set should be greater than the ratio of highest bandwidth and the lowest bandwidth.

Notice that the distributed feeding process is shared by all the peers within a cluster. Since the peers within every bandwidth class arrive in the system independently, we expect that the playback position of a peer is independent of its upload bandwidth. Hence, the average upload capacity of every cluster is equally likely. Therefore, heterogeneity does not cause any additional bottlenecks in the system.



(a) Homogeneous (b) Heterogeneous

Figure 11: Distribution of the achievable playback rate with heterogeneous upload capacity.

Figure 11 depicts the empirical pdf of the achievable playback rate[1] among regular peers (1000kbps, 80% of the peers) and low capacity peers (750kbps, 20% of the peers) in an heterogeneous setting. This shows that the performance of regular peers is not affected by the presence of low capacity peers: both the shape of the experimental pdf and the average of the achievable playback rate stay the same.

[1]Roughly speaking, the achievable playback rate is the maximum bitrate for the video file being downloaded so that it can be played without experiencing disruption. More details about evaluation metrics are given in Section 8.

7.4 Churn

Churn is normally characterized by the dynamic behavior of the peers. For example, a peer may join or leave in the middle of the video. Also, a peer may want to skip forward and backward in the video. Our objective here is to analyze the cost of allowing such operations and determining that our protocol can sustain such a behavior at a very high rate.

In order to join the system in the middle of the download a peer needs to download at least one piece in the most advanced segment, which is essentially the same as joining from the beginning. Further, a peer needs to construct a peer set such that it maintains the loosely coupled structure. This can easily be achieved in a distributed fashion if a peer connects to a peer within its current segment and then does a neighbor-of-neighbor search to construct its new peer set. Similarly, when a peer leaves in the middle of the download, all the links connecting to it are broken. However, these broken links will be quickly re-built since the peers periodically reconstruct their neighbor set. Finally, the procedure for skipping through the video is essentially similar to re-joining the video. We just argued that this is not an expensive process and can be achieved with negligible help from the tracker.

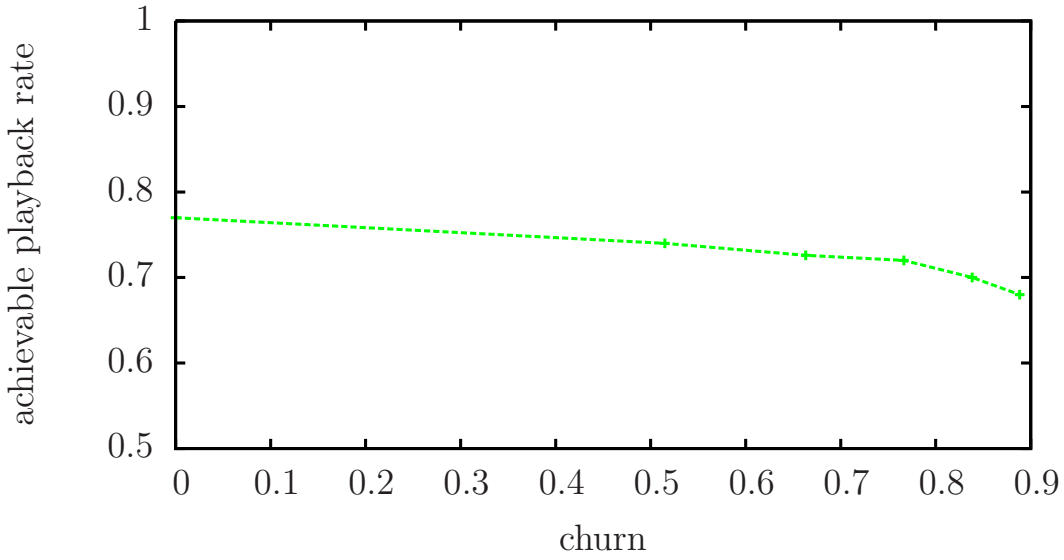


Figure 12: Average achievable playback rate in the presence of churn.

We evaluate here the performance of our hybrid solution in the presence of churn. Figure 12 plots the average achievable playback rate (as a fraction of the available upload bandwidth) when peers leave the system before the end of the download. At each round, a peer leaves with probability p_0. We characterize churn by the proportion of peers that leave the swarm during the progress of the download. Therefore, churn is quantified by $1 - (1 - p_0)^{\bar{T}}$, where $\bar{T}$ is the average number of rounds needed to download the whole movie. Figure 12 shows that the performance of our hybrid solution remains very good even for high values of churn. For instance, the average achievable playback rate decreases only by 9 points (from 77% to 68%) when 90% of the peers leave before the end of the download.

7.5 Free Riding

The tit-for-tat mechanism ensures that a peer can obtain new pieces only if it uploads pieces in return, making the protocol resilient to free riding. However, in order to bootstrap the protocol, when a new peer joins the system, BitTorrent uses an opportunistic unchoking mechanism allowing peers with low capacity to get pieces without uploading pieces in exchange. Although it increases the overall performance of the protocol, it also opens the door for free riders to download a file without uploading any content [13,19,21]. On the other hand, in our protocol no peer except the seed uploads pieces for free. Moreover, the seed only uploads pieces in the first and the last segment. In other words, peers cannot download the entire file only from the seed: they need help from other peers.

One may argue that a freerider may deviate from the protocol by downloading only pieces in its current segment to increase its immediate benefit, thus breaking the feeding process. First, such a behavior is not guaranteed to increase long term performance as the peer will quickly loose its bargaining power due to the lack of pieces in the future. Second, very efficient techniques using coding-based challenges have been proposed to force peers to download out-of-order pieces [5].

8 Evaluation

This section presents the results of extensive computer simulations. We compare the results obtained by our protocol with the previously known best results described in [2]. Section 8.1 describes the simulator and the set of parameters used for the simulations. Section 8.2 describes the performance metrics for which the algorithms are evaluated and Section 8.3 presents and analyzes the simulation results.

8.1 Experimental setup

We compare our algorithm with the unstructured algorithm presented in [2]. In order to establish connections, this algorithm uses random encounters where peers randomly poll the members of their peer set in order to establish a piece exchange. If both peers belong to the same segment, they try to exchange content within that segment. Otherwise, if they are in different segments then the less advanced peer can still download a piece in its current segment. However, the more advanced peer first tries to download a piece in its current segment, and if that fails, it tries to download any random piece available in the future.

We developed a discrete-time simulator similar to PeerSim [10], where time evolves in rounds. A peer is allowed to upload only a certain maximum number of pieces within a round. We also limit the number of pieces that can be downloaded within a round. However, the download limit is typically much higher than the upload limit. The tit-for-tat incentives are implemented at the round level, implying that a peer can download a piece from a neighbor only if it uploads a piece to that neighbor during the same round. Doing so ensures that peers contribute to the system as much as they get from it. Piece scheduling is performed by randomly picking two peers and initiating a bi-directional transfer between them. This stage is repeated until no pair of neighbors can exchange pieces. Note that two peers may exchange more than one piece with each other during one round. The neighborhood is built at every round in a random manner, matching the structures presented in previous sections.

The simulation results presented in the next sections have been obtained by running both algorithms in a network of peers joining the system at a rate of 5 peers per round (Poisson law). The neighborhood maximum size is set to 10 and the upload and download rates are set to 4 and 14 pieces per round, respectively. The structured algorithm splits the neighborhood of the peers in three sets: a set of peers in the same cluster (for swarming), a set of peers in the previous cluster and a set of peers in the next cluster. The size of the first set is limited to 6 and the sizes of the last two are each limited to 2. The system is seeded by a single seed with an upload bandwidth of 10. Note that due to the tit-for-tat incentives and the seeding capacity, a peer will always have enough download bandwidth capacity.

8.2 Evaluation metrics

We use three metrics to evaluate structured piece dissemination techniques compared to a purely random algorithm: (1) the fraction of upload bandwidth utilized for exchanging pieces (i.e., *throughput*), (2) the maximum rate at which the video can be played (referred to as *achievable playback rate*) and (3) the fraction of pieces downloaded in the current segment.

More specifically, the throughput is the number of pieces downloaded within a round divided by the maximum number that can be downloaded.

The achievable playback rate is the maximum rate at which the video can be played such that for every piece we have that it is available at the peer when its playback reaches the point where that piece is needed. Since a peer needs a setup time for buffering the first pieces of the video we allow a delay δ before starting playback. The achievable playback rate is given by the maximum rate r so that at any time t, all pieces up to $r \cdot (t - \delta)$ are available at the peer after t units of time as illustrated in Figure 13. For our simulations, the delay is set to the minimum required time to download two segments. Since

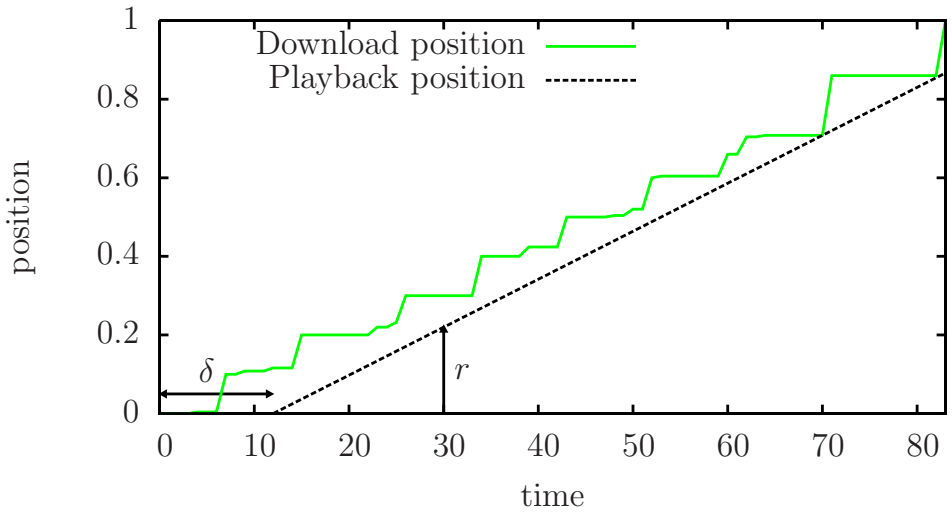


Figure 13: Evolution of playback position.

structured and random piece dissemination algorithms are equivalent in a flash crowd where almost all peers are in the same segment, we compare simulation results in steady state using a fixed rate Poisson peers arrivals scheme. Both throughput and playback rate are expressed as a fraction of the available upload bandwidth.

The fraction of sequential downloads refers to the fraction of pieces downloaded in the network that belong to the current segment of the downloading peer. This metric reflects the overall performance of the piece transfer algorithm and network topology in a VoD context.

To compare the performance of both piece dissemination algorithms in the context of a limited-memory device such as a set-top box, we run simulations of both algorithms with limited buffer size by making the peers drop pieces more than k segments in the past relative to the current playback position.

8.3 Experimental results

Next, we present results obtained by running 25 independent simulation instances each run over $2,000$ rounds. On average the achievable playback rate attained by structured dissemination is 77% of the upload bandwidth, whereas for the same set of parameters the playback rate attained by random dissemination as proposed in [2] is only 61%. Therefore, on average, structured dissemination is 16 points better than random dissemination. Figures 14(a) and 14(b) depict the empirical cumulative distribution function (cdf) and the empirical probability density function (pdf) for playback rate in steady state, respectively. It can be seen from Figure 14(a) that the playback rate of a peer under structured dissemination is stochastically larger than that under random dissemination. For example, the fraction of peers achieving a playback rate more than 0.68 under structured piece dissemination is over 93% whereas in random piece dissemination it is only 21%. In random dissemination, approximately 1% of the peers have a playback rate equal to zero, which implies that they are not able to even start the playback. An interesting observation is that the pdf of structured dissemination is narrower compared to that with random dissemination. This implies that there is more variance in the achievable playback rate under random dissemination.

There are two reasons that lead to the increase in the average achievable playback rate. First, the overall throughput in the system is increased due to better utilization of the piece diversity present in the swarm. Second, the goodput or the sequential throughput is also increased because the pieces are now disseminated along the linked list structure to maximize in order piece delivery. The average throughput achieved by random dissemination is 68%, whereas for structured dissemination it increases to 87%. Similarly, the average sequential throughput of the swarm under random dissemination is 66%, whereas for structured dissemination it is 75%.

Figure 15(a) and 15(b) depict the empirical cdf and the pdf, respectively, of the download rate. Notice that the peers using structured dissemination have a higher download rate. Also, the variance

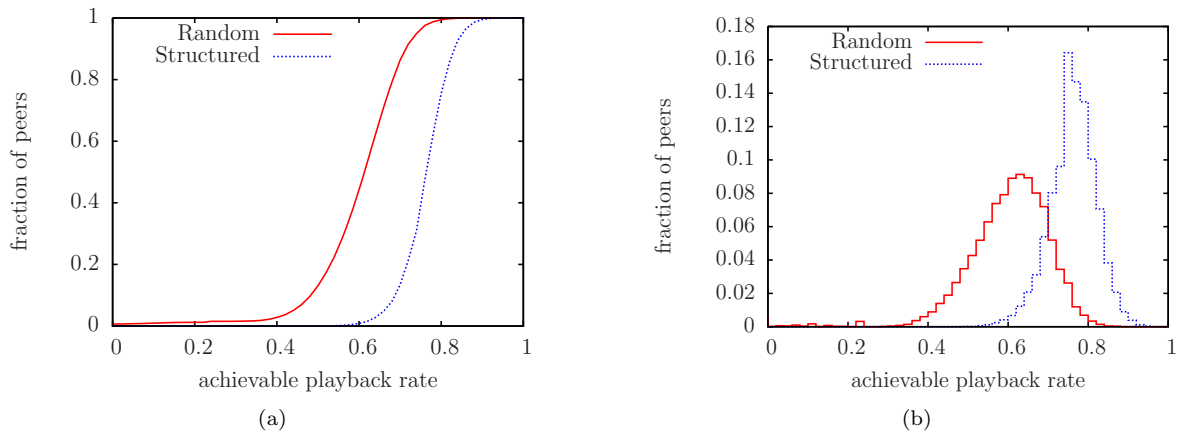


Figure 14: Experimental (a) cdf and (b) pdf of the achievable playback rate.

under structured dissemination is less, which implies a certain fairness in piece dissemination. This fairness can be attributed to the fact that piece delivery in a structured system is done over a linked list structure. Therefore, if there are adverse conditions in the swarm that result in reduced piece exchange, then all the peers are equally affected.

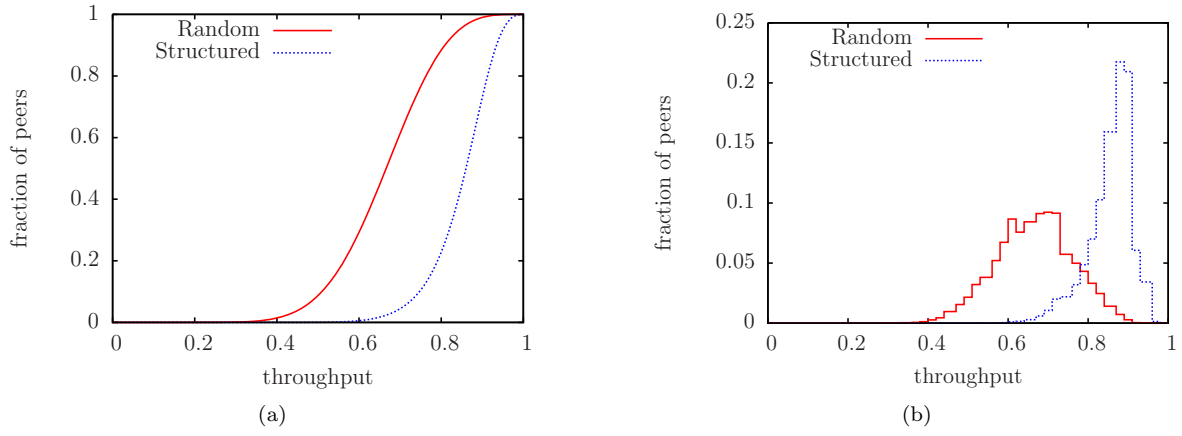


Figure 15: (a) cdf and (b) pdf of the download rate.

Figure 16 depicts the evolution of the number of peers in the system over the same set of parameters for both structured dissemination and random dissemination. According to Little's law from standard queuing theory, the reduction of service capacity of the system results in an increase in the number of customers queued in the system. Therefore, we expect the number of peers present in the swarm during structured dissemination to always be less than that during random dissemination. However, as seen in Figure 16, during the transient phase, towards the beginning of the process there are less peers departing from the swarm in structured dissemination compared to that in random dissemination. The reason is not necessarily the reduced throughput but the delay required in feeding the most advanced segment to the most advanced peers. However, in random dissemination there is no order in which pieces are seeded in the system. Therefore, when the first few peers are toward the end of their download, the most advanced pieces are missing in the system and they need to be uploaded by the seed. It takes a delay proportional to the length of the list before the pieces uploaded by the seed reach these departing peers. However, in case of random dissemination these pieces can be immediately pulled in by the most advanced peers. This slight blip in the performance should not be seen as drawback of the protocol but rather it shows that the most advanced peers are made to help the system by staying slightly longer to fulfill there commitment in the distributed feeding process.

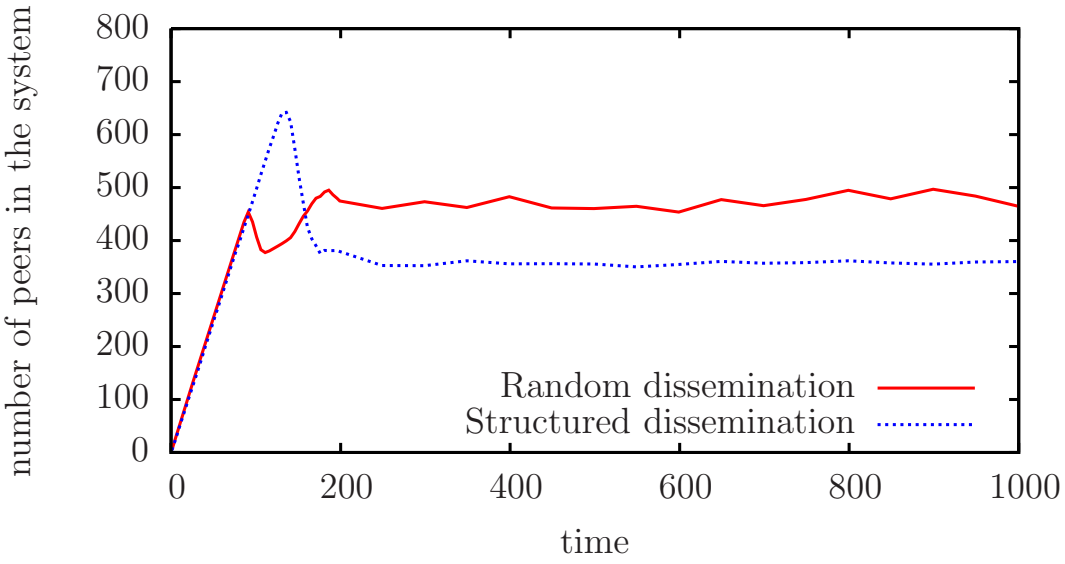


Figure 16: System size as a function of time.

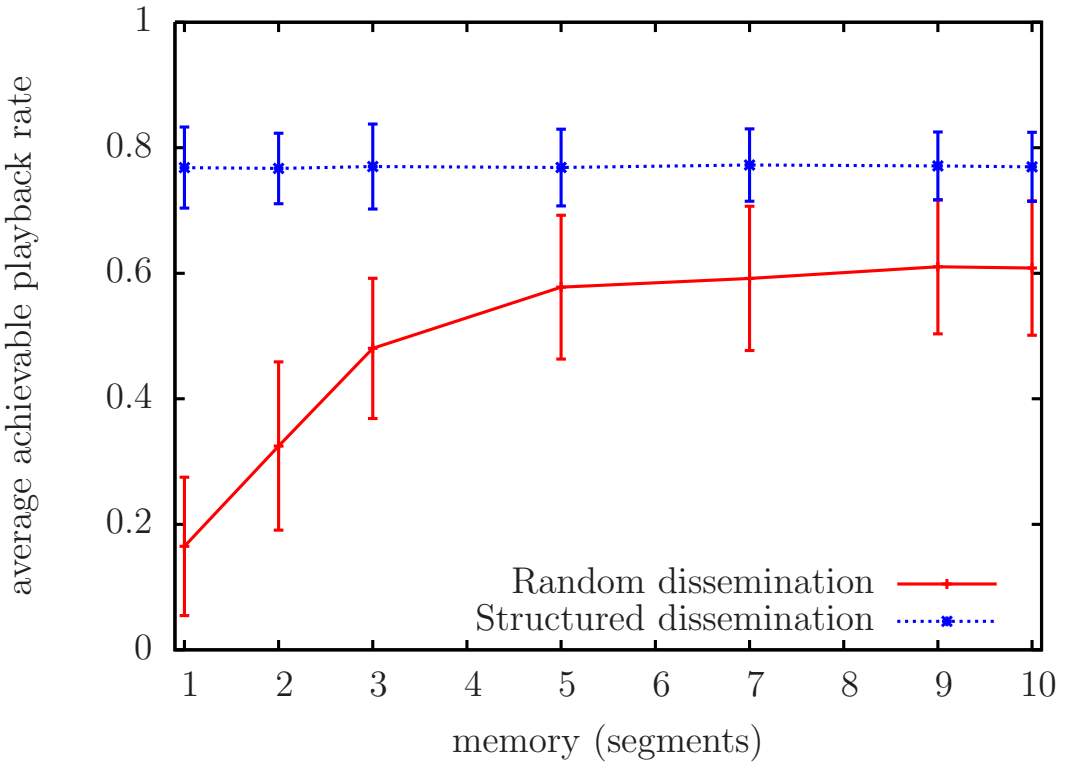


Figure 17: Structured dissemination vs. random dissemination with limited memory.

In Figure 17, we run the simulations for an application where memory size is constrained. Therefore, unlike before, a peer cannot store an infinite number of pieces but instead it keeps dropping the pieces which it has already viewed. Notice that random dissemination performs much worse compared to structured dissemination. The reasons for this performance improvement are intuitively clear as in structured piece dissemination a peer is allowed to communicate only within its cluster and the neighboring clusters. Therefore, there is no need to store more than one old segment.

9 Conclusions

Traditional file-swarming protocols are built using random graph structures. We demonstrate that an incentive-based VoD application built over a random structure cannot achieve optimal performance. Therefore, we propose disseminating content over a structured graph. Our solution is simple to deploy and is built in a decentralized fashion. We further demonstrate that the linked list is the most suitable structure for the VoD application. However, maintaining a linked list at the level of peers is a non-trivial task. Therefore, we introduce the concept of a cluster-level linked list, which allows us to obtain higher throughput within clusters of peers and at the same time a distributed feeding mechanism between two neighboring clusters. Using extensive simulations we show that the our proposal produces a significant improvement compared to the previously known best solution.

The authors in [1] have used network coding within segments to disseminate pieces. Notice that the network coding based optimizations can also be used in our algorithm. However, our goal in this paper is not to design an optimal solution, but instead we want to demonstrate that structured piece dissemination can achieve higher performance.

References

[1] S. Annapureddy, S. Guha, C. Gkantsidis, D. Gunawardena, and P. Rodriguez. Exploring VoD in P2P Swarming Systems. In *INFOCOM*, pages 2571–2575, 2007.

[2] S. Annapureddy, S. Guha, C. Gkantsidis, D. Gunawardena, and P. Rodriguez. Is High-Quality VoD Feasible Using P2P Swarming? In *WWW*, pages 903–912, 2007.

[3] N. Carlsson, D. L. Eager, and A. Mahanti. Peer-assisted On-demand Video Streaming with Selfish Peers. In *IFIP*, 2009.

[4] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-bandwidth Multicast in Cooperative Environments. In *SOSP*, pages 298–313, 2003.

[5] M.-L. Champel, A.-M. Kermarrec, and N. Le Scouarnec. Phosphite: Guaranteeing Out-of-Order Download in P2P Video on Demand. In *P2P*, 2009.

[6] Y.-H. Chu, S. G. Rao, and H. Zhang. A Case for End System Multicast. In *SIGMETRICS*, pages 1–12, 2000.

[7] B. Cohen. Bittorrent. http://www.bittorrent.com.

[8] B. Cohen. Incentives Build Robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer Systems*, 2003.

[9] Y. Guo, K. Suh, J. Kurose, and D. Towsley. P2Cast: Peer-to-peer Patching Scheme for VoD Service. In *WWW*, pages 301–309, 2003.

[10] M. Jelasity, A. Montresor, G. P. Jesi, and S. Voulgaris. The Peersim simulator. `http://peersim.sf.net`.

[11] L. Kleinrock. *Queueing Systems. Volume 1: Theory.* Wiley-Interscience, 1975.

[12] N. Laoutaris, P. Rodriguez, and L. Massoulie. ECHOS: Edge Capacity Hosting Overlays of Nano Data Centers. *SIGCOMM Computer Communication Review*, 38(1):51–54, 2008.

[13] T. Locher, P. Moor, S. Schmid, and R. Wattenhofer. Free Riding in BitTorrent is Cheap. In *HotNets*, 2006.

[14] N. Magharei and R. Rejaie. PRIME: Peer-to-Peer Receiver-driven MEsh-Based Streaming. In *INFOCOM*, pages 1415–1423, 2007.

[15] N. Magharei, R. Rejaie, and Y. Guo. Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches. In *INFOCOM*, pages 1424–1432, 2007.

[16] J. Mol, J. Pouwelse, M. Meulpolder, D. Epema, and H. Sips. Give-to-Get: Free-riding-resilient Video-on-Demand in P2P Systems. In *MMCN*, 2008.

[17] N. Parvez, C. Williamson, A. Mahanti, and N. Carlsson. Analysis of BitTorrent-like Protocols for On-Demand Stored Media Streaming. *SIGMETRICS*, 36(1):301–312, 2008.

[18] F. Pianese, D. Perino, J. Keller, and E. Biersack. PULSE: An Adaptive, Incentive-Based, Unstructured P2P Live Streaming System. *IEEE Transactions on Multimedia*, 9(8):1645–1660, Dec. 2007.

[19] M. Piatek, T. Isdal, T. E. Anderson, A. Krishnamurthy, and A. Venkataramani. Do Incentives Build Robustness in BitTorrent? In *NSDI*, 2007.

[20] F. Picconi and L. Massoulié. Is there a future for mesh-based live video streaming? In *P2P*, pages 289–298, 2008.

[21] M. Sirivianos, J. H. Park, R. Chen, and X. Yang. Free-riding in BitTorrent with the Large View Exploit. In *IPTPS*, 2007.

[22] V. Valancius, N. Laoutaris, L. Massoulie, C. Diot, and P. Rodriguez. Greening the Internet with Nano Data Centers. In *CoNEXT*, 2009.

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399