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

Kévin Huguenin, Anne-Marie Kermarrec
IRISA / INRIA
Rennes, France

Vivek Rai, Maarten van Steen
Vrije Universiteit
Amsterdam, The Netherlands

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. In VoD applications pieces are downloaded in order. This prevents us from directly applying a BitTorrent-like tit-for-tat incentive scheme. We advocate the use of a loose structure in P2P VoD applications to achieve high playback rates. In this paper we propose a decentralized piece dissemination scheme built on loosely coupled structures maintained using gossip. Peers are grouped into clusters depending on their playback position. Swarming is performed within 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% to 77% the achievable playback rate.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed Applications*

General Terms

Algorithms, Design

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 broadcast content and the quality of service requirements of VoD applications make centralized solutions costly.

Over the last decade, the peer-to-peer (P2P) paradigm proved to be an efficient way to distribute content in a decentralized fashion [2,9]. A very popular P2P swarming protocol enabling file sharing between peers is BitTorrent [4]. 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 drastically decrease 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 complete 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 that a peer contributes to the swarming process as much as it exploits the system. 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 to one other?

In this paper we propose 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, there is a set of swarming

NOSSDAV'10, June 2–4, 2010, Amsterdam, The Netherlands.

subsystems, referred to as *clusters*, which enable peers close with respect to playback position, to exchange pieces of immediate interest as in traditional swarming systems. 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 bargaining power.

We evaluate our solution through simulations on top of the BitTorrent framework. The protocol consistently outperforms the tit-for-tat-compatible VoD system based on random exchanges proposed in [1]. Typically our structured piece dissemination protocol improves the achievable playback rate from 61% to 77% and the throughput from 68% to 87%. Moreover, our simulations show that, in constrast to an unstructured piece dissemination protocol, the efficiency of 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 resource-constrained devices such as set-top boxes (for instance some IPTV boxes have only 1GB of memory while a high-definition video file is of a few gigabytes). When nodes are allowed to store only 10% of the whole file (slidding window), our protocol achieves a playback rate of 77% whereas the unstructured protocol achieves a playback rate of only 18%.

Section 2 provides relevant work in the design of peer-to-peer VoD systems. Section 3 gives our design rationale. Section 4 provides a high-level description of our algorithm, followed by a detailed description in Section 5. There, we also show how to provide a fully decentralized implementation. Protocol analysis and simulation results are given in Section 6 and Section 7. Section 8 concludes the paper.

2. RELATED WORK

Most of the large-scale peer-to-peer content dissemination schemes are designed using a (multi)tree-like structure [2,5]. A general argument against such approaches is their relatively higher cost of maintenance in a dynamic environment [7]. 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 [4] have also been proposed for content dissemination [6]. The advantage of such a solution is high scalability due to decentralization. In BitTorrent-styled file dissemination, a file is divided into multiple pieces such that each piece is independently downloaded. The incentive mechanism, namely tit-for-tat, implies that in order to download a piece, a peer must upload a piece in return. Tit-for-tat cannot be used directly in VoD applications where the piece diversity is low and two peers at different playback position have no mutual interest in exchanging pieces.

A solution introduced in [8] 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. Random downloads help increase the throughput of the swarming process, while downloading pieces in a sequential order is critical for the performance of the VoD application.

Increasing piece diversity can also be achieved by slightly relaxing the sequentiality requirement using the segment model introduced in [1]. In this model, a file is divided into nonoverlapping segments, where a segment is a group of continuous pieces. The segments are downloaded in a sequential order, but 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.

Note 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. 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. Hence piece exchange is not possible. In order to make possible an exchange between the 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. Note that since, in BitTorrent, a piece can only be uploaded by a peer after it has been completely and successfully received (i.e., full download requirement), this approach works only if all peers are at different playback positions.

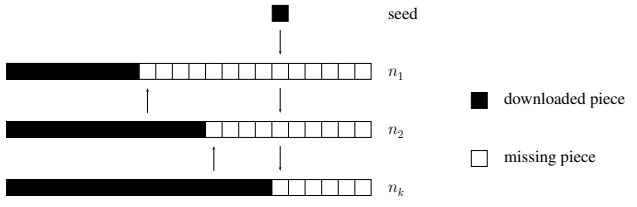


Figure 1: The seed forwards the most advanced piece to the least advanced peer. This piece is eventually uploaded to the most advanced peer on a forward path. In return, in order pieces are downloaded on a reverse path.

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. However, this approach poses several problems: *(i)* only half of the upload bandwidth used serves the peers' immediate interests; *(ii)* it requires all peers to be at a different playback position; *(iii)* the throughput of the linked list is limited by the bandwidth of the slowest peer; *(iv)* it is hard to maintain such a structure in practical decentralized setting in the presence of churn.

4. 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, we concluded 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 2. The seeding capacity can be equally divided between all the linked lists. However, this solution is not scalable either because the number of linked lists grows with the size of the system. this solution also suffers from a poor performance due to lack of piece exchange between peers at the same playback position. In addition, it is based on a fragile structure hardly maintainable in a practical setting.

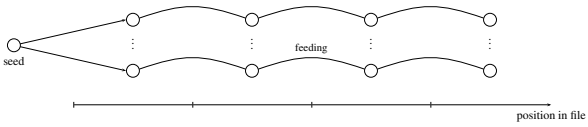


Figure 2: Distributed feeding using multiple linked lists at the peer level (horizontal piece exchanges)

Remember that the solution proposed in [1] 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. Thus, 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, 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 3. However, if the number of clusters is very high, the available seeding capacity may not be sufficient.

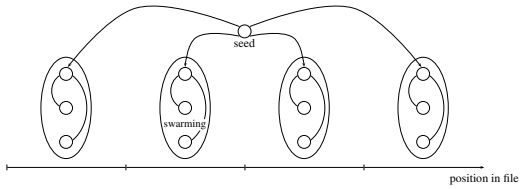


Figure 3: Independent swarming with centralized seeding (vertical piece exchanges)

Notice that the distributed feeding technique as depicted in Figure 2, which is based on horizontal piece exchanges is complementary to the independent swarming technique shown in Figure 3 that is based on vertical transfers. However, neither of these 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 4). In order to achieve that, we allow the peers within the same segment to be grouped 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.

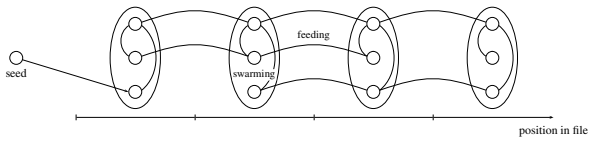


Figure 4: Hybrid solution at a cluster level

There are several advantages to this hybrid scheme, for example, the length of the list is now solely 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 above. Note that even in this model, the horizontal piece exchanges are done over a linked list. However, these linked lists are not independent and they can thus 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%. Nevertheless, in the cluster model, all peers within a cluster are responsible for feeding pieces. Thus, the fraction of bandwidth utilized per peer in feeding pieces is significantly reduced and hence a *much higher* goodput can be expected.

5. ALGORITHM DETAILS

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. 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. Sec-

ond, we introduce a management technique for a peer set (i.e., the set of peers a node can exchange pieces with) 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. Eventually these pieces reach the most advanced cluster. A pseudo-code version of the seeding policy is given by Algorithm 1. 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.

Algorithm 1 Seeding policy

Input Seeding
  s: seeding capacity
  S^+ and S^-: most and 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

Peer Set Management The peers in 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. When a new peer joins the swarm it gets connected to some peers within the first group. The remainder of the peer set is constructed by connecting to neighbors of neighbors. The structure is maintained during the download through gossip: peer set is updated periodically by exchanging set of neighbors with current neighbors similarly to [10]. 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 using gossip. This way we can easily maintain the structure in a decentralized way.

Exchange policy The exchange policy determines whether two peers n_1 and n_2 should exchange pieces 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 be pushed only 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 the 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 in turn a long reverse path establishing 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.

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 (resp S_2): n_1's (resp. n_2'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

Note that using the transfer strategy presented in the previous paragraph, pieces before a node's playback position are used only to feed peers in the previous group. Therefore, a peer can drop pieces before the playback position of the peers in the previous group without reducing its feeding ability.

6. DISCUSSION

In this section, we analyze and discuss the behavior and performance of our protocol in face of traditional issues specific to large-scale peer-to-peer systems deployed in the public domain.

Protocol Stability First we consider the stability of the

protocol. By protocol stability we mean whether the system will keep on functioning 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 is blocked, 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*. Effectively, 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.

Bottlenecks and heterogeneity We now determine that the bottlenecks we identified in the peer-level linked list can be easily prevented in the cluster-level linked list. A bottleneck can occur if the feeding capacity falls below a certain critical value, which results in all subsequent clusters getting starved. Therefore, adjusting the size of the clusters (taking into account the *average* upload bandwidth of peers) so that they able to feed an entire segment in one round ensures 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. To ensure that this property is satisfied, one can either set the number of clusters such that it is highly unlikely that the size of the cluster drops below the critical value or, if the system size is not known in advance, the size of the clusters can be automatically adjusted by merging two consecutive clusters, whenever the size of one of the two clusters drops below the critical value.

Churn and VCR operations We analyze here the cost of peers leaving and joining the system in the middle of the download. To do so, a peer needs to download 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 is automatically achieved by the gossip-based maintenance technique that leverages neighbors-of-neighbors. 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. Finally, the procedure for skipping through the video is essentially similar to re-joining in the middle the video.

Free Riding The tit-for-tat mechanism ensures that a peer obtains new pieces only if it uploads pieces in return, making the protocol resilient to free riding. However, one might 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 [3].

7. EVALUATION

This section presents the results of our simulations.

7.1 Experimental setup

We compare our algorithm with the unstructured algorithm presented in [1]. 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, where time evolves in rounds. A peer is allowed to upload only a certain maximum number of pieces within a round. 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.

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 media file has been split in 10 segments of 25 pieces. The peer set size is set to 10 and the upload limit is set to 4 pieces per round. The system is seeded by a single source with an upload bandwidth of 10.

7.2 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.

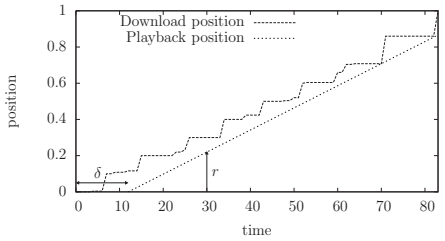


Figure 5: Evolution of playback position.

The achievable playback rate is the maximum rate at which the video can be played: this means that whenever a peer reaches a given playback position, the corresponding piece is available. Since a peer needs a setup time for buffering the first pieces of the video, we allow a delay δ before

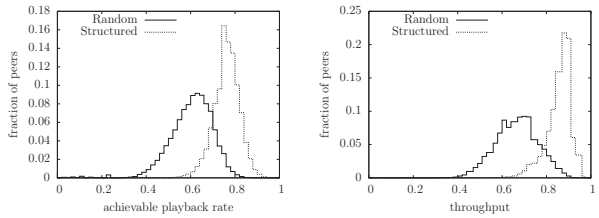
starting playback. The achievable playback rate is given by the maximum rate r so that at any time t, all pieces up to $r \cdot (t - \delta)$ are available at the peer after t units of time as illustrated in Figure 5. The delay is set to the minimum required time to download two segments. Since structured and random piece dissemination algorithms are equivalent in a flash crowd scenario where almost all peers are in the same segment, we compare simulation results in steady state using a fixed rate Poisson peers arrival 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, 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.

7.3 Experimental results

The following results are 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 [1] is only 61%. An achievable playback rate of 77% means that a peer with an upload bandwidth of 1Mbps can smoothly watch a stream of 770kbps. Figure 6 depicts the empirical probability density function (pdf) for throughput and playback rate. 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.



(a) achievable playback rate (b) throughput

Figure 6: Experimental pdfs.

The average achievable playback rate is better for two reasons. 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%, and 87% for structured dissemination. Similarly, the average sequential throughput under random dissemination is 66% and 75% for structured dissemination.

In Figure 7, we constrain the memory size to a fixed number of segment: a peer cannot store an infinite number of pieces but instead it keeps dropping the pieces which it has already viewed. We observe that structured dissemination performs drastically better compared to random dissemination. The reasons for this performance improvement are intuitively clear as in structured piece dissemination a peer is allowed to exchange pieces only within its cluster and the neighboring clusters. Therefore, there is no need to store more than one old segment.

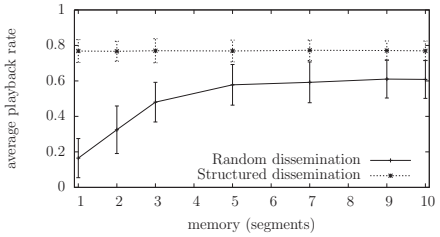


Figure 7: Structured dissemination v.s. random dissemination with limited memory.

8. CONCLUSIONS

Traditional file-swarming protocols are built using random graph structures. We advocate the use of content disseminating over a loosely structured graph to achieve higher performance. We introduce the concept of a cluster-level linked list. Our hybrid solution allows us to obtain higher throughput within clusters (vertical dissemination via traditional swarming) and at the same time an efficient distributed feeding mechanism between neighboring clusters (structured horizontal dissemination). Our solution is practical, simple to deploy and is built in a fully decentralized fashion. Using simulations, we show that the our proposal produces a significant improvement compared to the previously known best solution.

9. REFERENCES

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

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

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

[4] B. Cohen. BitTorrent. http://www.bittorrent.com.

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

[6] N. Magharei and R. Rejaie. PRIME: Peer-to-Peer Receiver-driven MEsh-Based Streaming. In *INFOCOM*, 2007.

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

[8] N. Parvez, C. Williamson, A. Mahanti, and N. Carlsson. Analysis of BitTorrent-like Protocols for On-Demand Stored Media Streaming. In *SIGMETRICS*, 2008.

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

[10] X. Qiu, C. Wu, X. Lin, and F. C. Lau. InstantLeap: Fast Neighbor Discovery in P2P VoD Streaming. In *NOSSDAV*, 2009.