On Some Fundamental Properties of P2P Push/Pull Protocols

(Invited Paper)

Renato Lo Cigno, Alessandro Russo
DISI – Univ. of Trento
Trento, Italy
{locigno,russo}@disi.unitn.it

Damiano Carra
Institut EURECOM
Sophia Antipolis, France
carra@eurecom.fr

***Abstract*—The peer-to-peer communication paradigm is changing the way the Internet works, and the perspective network and service providers are looking at the telecommunication business. P2P applications and networks are taking foot by the day, and new systems are proposed continuously with ever novel features and better performances. In spite of attention and success, however, there is still a lack of fundamental analysis and understanding of the elementary properties of these systems. In this paper we consider a class of P2P protocols suitable both for content delivery (file-based communications) and for high-bandwidth media streaming like video and TV, and explore its fundamental properties. The class considered is known as mesh-based swarming push-pull systems or *interleave* protocols. They split the content in pieces and alternate continuously two phases: One where the peer pushes a piece to another peer to percolate information in the system, and the other when it pulls a piece trying to retrieve missing information. We compare synchronous and asynchronous models and explore the impact of protocols parameters, such as the dimension of the active neighborhood, trying to identify the efficiency of these very simple protocols in different scenarios, gaining insight to design the next protocol generation with performance and efficiency in mind.**

***Index Terms*—P2P networks; Overlay; Information Exchange; Streaming; Chunk-Based Distribution; Simulation; Distributed Systems; Push/Pull Protocols.**

I. Introduction

The rise of Peer-to-Peer (P2P) communication paradigm seems to be the next big thing in Internet services. After the initial phase, where legal concerns about illegal exchanged content hampered the diffusion, the key idea of sharing resources to improve the common benefit of users is gaining momentum, and service providers are modifying their business plans and strategies to ride the rising tide.

After the emergence (and often the fall) of early file-sharing P2P systems like Napster [1], Gnutella [2], Kazaa [3], and others, different applications were supported by P2P systems, starting from voice, (Skype being the most famous example) to content delivery and video. The growth of legal and commercial services on P2P systems is also rising the requirements for performance, resilience and always new, more appealing systems. To achieve these requirements, better understanding of existing systems is needed and new methods and algorithms are called for. In early performance works [4][5][6], we proposed new, semi-analytic techniques to explore fundamental properties of the topology used by P2P overlays as well as the the protocols used to build them. In this work we focus instead on the information exchange process, and use simulations with different level of abstraction to investigate fundamental properties that drive the diffusion of the information within a mesh-based swarming system.

The type of information that is exchanged within the P2P system normally does not influence its behavior and performance. There are however a few key points that need precise definition, and different applications that have different requirements in transferring the information. Application can be file-based or stream-based. In file-based applications the service can be used only when the whole file is transfered: Average throughput and total transfer delay are the key performance parameters. In the stream-based case the service is used alongside the information transfer: Minimum throughput and latency from the information generation to its fruition are the main evaluation variables. Additionally, the information can be split for transmission in stand-alone, autonomous pieces normally called *pieces* or chunks, or can be organized in one or more fluid-like flows of single packets. In this work we concentrate on chunk-based transmission, both for file-based and stream-based systems.

Information diffusion in chunk-based P2P overlays can be classified into three main classes:

Push: Pieces are *passed down* from one peer (parent) to another (child). The parent don't ask the child which piece to push, so in presence of multiple parents and non-structured systems, multiple pieces can reach the same peer and some others never arrive, leading to inefficiencies. Push is normally and naturally associated to structured, tree-based topologies [8][9][10][11][12][13], in case of long-lasting neighborhood relations pushing is very efficient.

Pull: Pull is the opposite of push: One peer (child) ask for one piece to another (parent) without knowing whether the parent has the piece or not. Duplication is not a problem, but starvation may occur because a child cannot find any parent with the piece it is

This work was supported by the Italian Ministry of University and Research (MiUR), with the Grant PRIN-2006099023 "Profiles" (disi.unitn.it/profiles)

looking for. Pull systems are normally associated with swarming, unstructured systems, where peers seek for multiple parents, balancing the starvation probability with redundant neighborhood searches. Practical systems actually let nodes to exchange information about the pieces they have, thus pure pull systems are used mainly as a reference model.

Status-based: These systems are based on extensive signaling between peers that exchange status tables (e.g., the bitmap of a video playout buffer: 0 for a missing chunk, 1 for an available one) describing the download process. Piece transfers are negotiated among peers [14][15][16][17].

Push and Pull systems have the appeal of simplicity: No signaling is required to coordinate peers and thus they became naturally very resilient to churn and other impairments. Status-based systems instead seek for performance and efficiency trading in simplicity and the risk of becoming fragile and prone to failure in face of heterogeneity in the network and random behavior of peers.

An interesting approach is represented by the recently proposed Interleave [7]. The scheme combines push and pull with a particular piece selection policy (for details, see Sect. II), yet maintaining no exchange of update information about pieces owned by nodes.

There are few other papers that propose a combination of pull and push [18][19]. Most of them use the push mechanism for spreading rapidly the content, and the pull mechanisms for filling the holes in the received stream or to subscribe to different trees in case of a multiple-tree structure. In any case, no proposal considers to *interleave* the pull and push mechanisms.

Interleave was designed mainly for P2P file transfer. The analysis of the scheme made in [7] was done considering that all the nodes in the network are homogeneous and synchronized, i.e. the download time of a single piece is the same for all nodes and the time is slotted. Authors of [7] found that the file dissemination time of Interleave is within a constant factor of the optimal performance of a fully centralized system. A major concern is whether the good performance of Interleave is due to synchronization and bandwidth homogeneity or not. Also, while [7] speculates that Interleave may be used for live streaming, it does not further investigate this issue.

In this work we analyze in detail the performance of the Interleave scheme: In particular we "translate" the scheme into a fully distributed protocol, where nodes in the network behave independently, without coordination.

The scheme can be used for file distribution, as well as for live streaming. In this case, the correct receiving order of the pieces and the delay of each piece represent the main challenge. The results show that the protocol we propose is able to maintain the performance of the synchronized Interleave scheme in case of file distribution; moreover, it is able to provide delay bounds in case of live streaming. To the best of our knowledge it has never been shown that a scheme as simple as Interleave, which is a combination of pull and push and it does not maintain information about the pieces owned by nodes, can already provide such a good performance.

The remaining part of the paper is organized as follows. Sect. II describes the fundamental workout of push/pull protocols, both in a simplified, cycle-based model and in a more realistic, entirely distributed system. Sect. III discusses numerical results and fundamental system parameters. Sect. IV closes the paper with conclusions and future work.

II. Interleave Protocols

We consider a system with a single source, where the content is partitioned into pieces that can be exchanged independently. Pieces are generated at a constant rate r_{str}, which can be the streaming rate or simply a service rate for a file transfer. Each piece has a sequence number that reflects the order of creation by the source. Each node *alternates* between pull and push mode, and has a finite size neighborhood defined by its contact list of size k; the neighborhood of P is defined as the set of peers that can be contacted *actively* by peer P, i.e., P can contact peers only in its contact list, but can be contacted by peers that are outside its contact list.

In *push mode*, a node P randomly selects a neighbor and a piece to push; if the neighbor does not have the piece and has available download bandwidth, P uploads the piece, otherwise the push is aborted.

In *pull mode*, a node P randomly selects a neighbor and a piece and sends a pull request to that neighbor. If the neighbor has the requested piece and it is currently not uploading to any other node, it accepts the request, otherwise it refuses it.

The piece selection policy represents a delicate part of the design, especially for streaming systems, where each piece should be received within a maximum delay. A random piece selection may work for file distribution, but not for live streaming. Also, 'intelligent' selection procedures, like rarest-first and similar, cannot be implemented without state signaling between peers, thus cannot be implemented in a basic push/pull protocol without state. We adopt the same piece selection policy proposed in Interleave [7]:

- In push mode, the node pushes the piece with the highest sequence number among the pieces received *via a push* from one of its neighbors.
- In pull mode, the node asks for the piece with the *lowest sequence number it does not possess*. This policy aims to fill the holes within the sequence of pieces.

Interesting features to be explored in the future, include simple choice-based selection policies (e.g., instead of pushing the youngest piece, offer a choice or more pieces and let the destination peer select among them) ad similar strategies that maintain the system without state, thus highly dynamic and resistant to churn.

A. A Cycle-based Model

The basic scheme described above was initially proposed considering a simplified environment, where all the nodes have the same upload link bandwidth and unlimited download link

bandwidth. It is also assumed that the time to upload a single piece is much bigger than the latency for the exchange of a single (application level) message. Additionally it is supposed that the nodes are synchronized and the time slotted. In even slots all the nodes are in push mode; in odd slots all the nodes are in pull mode.

If the node is in push (pull) mode and the request is accepted by the selected neighbor, the piece is pushed (pulled). At the end of the slot the node switches mode. If the request is refused by the selected neighbor, the slot is entirely wasted, and the node waits until the end of the slot before switching mode (and sending a new request).

The source pushes a new piece in every even slot, and it replies to pull in odd slots. This means that the streaming rate is equal to one piece every two cycles and the required upload bandwidth is $2r_{\text{str}}$.

This cycle-based model was used in [7] to find theoretical bounds for the distribution process, and we use it for comparison reasons. In a realistic scenario, upload link bandwidths are heterogeneous, downlink capacity is not infinite, and imposing synchronization may be impractical (if at all possible), thus a better model is required to understand fundamental properties.

B. A Realistic Model

In a real distributed system, the behavior of each node is independent from all other nodes. Nodes are desynchronized, because forcing synchronization is costly and can also lead to high inefficiency (synchronization can be based only on the least performing peer). A node switches from one mode to the other (e.g., from push to pull) either after a request is accepted and the corresponding piece transfer is finished, or after receiving a maximum number of refusals to requests (at each try, a new neighbor is randomly selected). This behavior keeps the system running smoothly and avoid starvation and blocking.

The time spent in pull or push mode depends on the success of the requests and on the availability of the bandwidth (both, the neighbors' and the node's bandwidth); thus each push or pull interval is not fixed and the nodes, even in a homogeneous case, will be desynchronized.

We stress that this implementation is entirely decentralized and the information exchange among peers is kept to a minimal level: Keep-alive messages for the contact list management and push/pull queries and answers. Additionally, the goal of this work is not building a new system (not yet at least!), but to gain insight into fundamental parameters like the contact list size, the efficiency of the distribution system (i.e., the ratio between the available bandwidth in upload/download and the streaming or transfer rate).

Algorithm 1 summarizes the basic operations of a single node related to *sending out requests*[1]; Algorithm 2 summarizes the basic operations related to replying to requests. We do not report the pseudo-code for building and maintaining the contact list for the sake of brevity.

Notice that these simple algorithms generate asynchronous behaviors even within the same node. Indeed, a node controls its own alternating between push and pull in sending out requests, but has no control on the requests he receives, thus a node can be in one of four states, depending on its *active/passive* status (not considering periods of idle behavior during the signaling related to requests' negotiation): Pushing/pulled, pulling/pushed, pushing/pushed and pulling/pulled. We use the desinence -ing to identify the active status consequent to sending out a request, and the desinence -ed to identify the passive status of answering a request. Notice that in the two states pushing/pulled and pulling/pushed, the two coexisting transmissions compete for the same uplink (downlink respectively) resources.

Algorithm 1: Pseudo-Code of Interleave: Sending Out Requests

```
Input: Maximum number of push and pull attempts:
       max_push_attempts, max_pull_attempts)
if (status == PULL) then
    while (pull_attempt < max_pull_attempts) AND (download
    bandwidth available) do
        Select a neighbor and the lowest pieceID not owned;
        Send PULL message;
        Wait for reply;
        if (reply == ACCEPT_PULL) then
            if (download bandwidth available) then
                Send READY message;
                startPullingPiece;
                break; /* exit from the while cycle  */
            else
                Send BUSY message;
            end
        end
        pull_attempt++;
    end
    status = PUSH;
    pull_attempt = 0;
end
if (status == PUSH) then
    while (push_attempt < max_push_attempts) AND (upload
    bandwidth available) do
        Select a neighbor and the highest pieceID owned;
        Send PUSH message;
        Wait for reply;
        if (reply == ACCEPT_PUSH) then
            if (upload bandwidth available) then
                Send READY message;
                startPushingPiece;
                break; /* exit from the while cycle  */
            else
                Send BUSY message;
            end
        end
        push_attempt++;
    end
    status = PULL;
    push_attempt = 0;
end
```

III. Numerical Results

We have implemented both the cycle-based and the more realistic, asynchronous and distributed model in a simulation environment, to explore some fundamental parameters. The cycle-based implementation works in the same ideal hypotheses of the theoretical analysis in [7] as summarized in

[1]For clarity we have not reported here the full algorithm, where we consider also the case when a neighbor does not reply to a request (we use a timeout mechanism to manage these situations).

Algorithm 2: Pseudo-Code of Interleave: Replying to Requests

```
if (message == PULL) then
    if (pieceID available) AND (upload bandwidth available) then
        send ACCEPT_PULL message;
        wait for reply;
        if (reply == READY) then
            startPieceUploading;
        end
    else
        send REFUSE_PULL message;
    end
end
if (message == PUSH) then
    if (pieceID missing) AND (download bandwidth available) then
        send ACCEPT_PUSH) message;
        wait reply;
        if (reply == READY) then
            startPieceDownloading;
        end
    else
        send REFUSE_PUSH message;
    end
end
```

Sect. II-A. The asynchronous model is based on the protocol described in Sect. II-B, along with Algorithms 1 and 2. It includes the signaling for pieces exchange and some simple models of the underlying information transport network. The actual transmission of pieces, e.g., with TCP/IP, is not implemented since is would slow down simulation and make the results so complex as to make their interpretation almost impossible.

A. The PeerSim Environment

PeerSim is a Java based simulator that consists of many configurable components: It has two types of engines, *cycle-based* and *event-based*, and different modules that manage the overlay building process and the transport characteristics. In the *cycle-based* engine, all nodes are synchronized, making it a perfect tool for the implementation of the simple cycle-based model. In each cycle each node is activated sequentially and executes a protocol: There is no concurrency among nodes nor competition for resources. In the *event-based* engine, all nodes are independent and run concurrently (they are independent instances of the same class). They can add events in the event list of the simulator, so nodes can compete for resources.

With PeerSim it is possible to build different network overlays; the models for some of them (e.g., random graph) are present in the simulator. It is also possible to specify if the edges of the overlay graph are directed or not. With random graphs, the degree of each node is random, because it depends on how many incoming edges have been created. We create a model with constant-degree, where all the nodes has the same number of edges (and neighbors). This overlay is similar in spirit to the one built by BitTorrent.

The simulator contains a transport layer for sending messages from a source to a destination: The layer adds a delay uniformly distributed between a minimum and a maximum value an it can also drop messages with a probability p.

For a detailed description of PeerSim simulator the interested reader is referred to [20].

B. Parameters under Study and Settings

The theoretical analysis of the basic Interleave scheme in [7] is done considering a directed random graph with degree k. Each peer selects its own k neighbors independently from any other peers and without constraints. Each peer has a contact list of exactly k neighbors, and the probability of being in the contact list of other nodes is binomially distributed with mean k. We refer to this overlay as *asymmetric*.

We also define and test another overlay model, where the the edges are not directed: In this case we use a constant-degree graph with degree k. Building it is somewhat more complex, since the choice of neighbors to build the contact list, requires signaling and coordination and can be difficult if the contact list is large and the number of peers small. We refer to this overlay as *symmetric*.

The streaming rate of the source, r_{str}, is set to one piece per second. In the cycle-based model this implies that each cycle is equal to 0.5 s. In the event-based case, we set the piece size to 15.625 kbytes (125 kbit). So, the minimum bandwidth assumed for the uplink (128 kbit/s) is sufficient to leave some capacity for maintenance and piece request messaging.

We start considering a homogeneous case, where all nodes have the same bandwidth, and we test three different homogeneous bandwidths: 256 kbit/s and 512 kbit/s. Note that, with a bandwidth of 256 kbit/s, the source can serve the each piece twice, because the streaming rate is one piece per second. Thus this case corresponds to the cycle-based case, where a new piece is generated every even cycle and served again in odd cycle. If not otherwise stated, each node has a limited download bandwidth equal to four times the upload bandwidth.

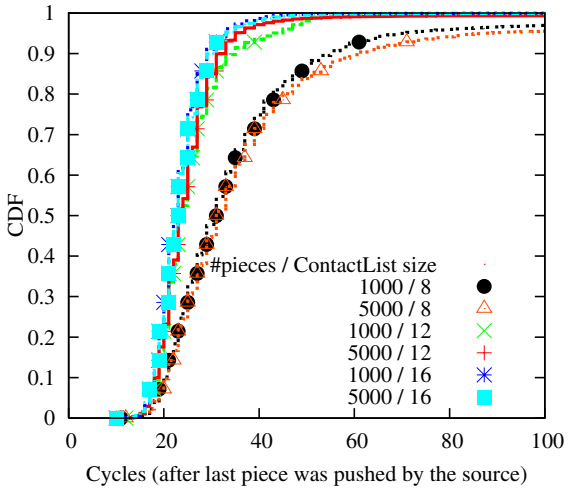
We analyzed many scenarios with different network sizes and number of pieces. In particular, the number of nodes N can be equal to 100, 500 and 1000. The number of pieces C can be equal to 10^3, $5 \cdot 10^3$ and 10^4.

In order to evaluate the performance of different models with different parameter settings, we consider three main performance indexes:

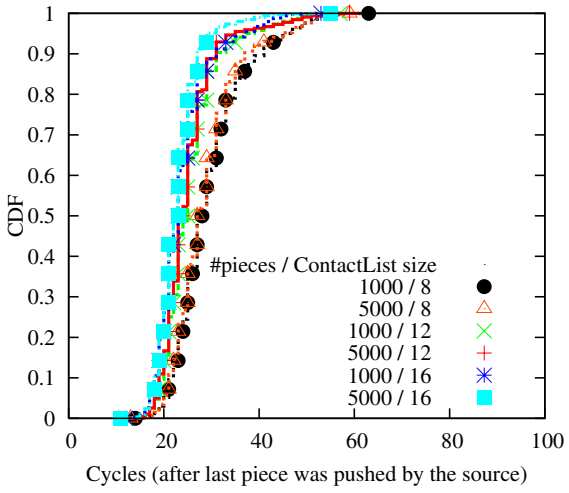
- **Completion Time:** In case of file distribution, the main performance metric is the time at which all nodes receive all pieces.
- **Maximum Delay:** In case of live streaming, the delay of each piece represent the performance index of primary interest. We compute it considering all the nodes and all the pieces. Piece are numbered $0, \ldots, C$. Piece c is produced at time t_c, and it reaches the node P at time $t_{c,P}$. The maximum delay, $d^{\max}$, is the maximum over all pieces and over all nodes:

$$d^{\max} = \max_{c,P}(t_{c,P} - t_c) \ .$$

- **Number of Operations:** Since each node alternates between push and pull, it is important to understand how much efficient are these two mechanisms; we compute the number of pieces retrieved via pull and the number of pieces received via push.



(a) Asymmetric contact list



(b) Symmetric contact list

Fig. 1. CDF of the overall download time with $N = 1000$ nodes.

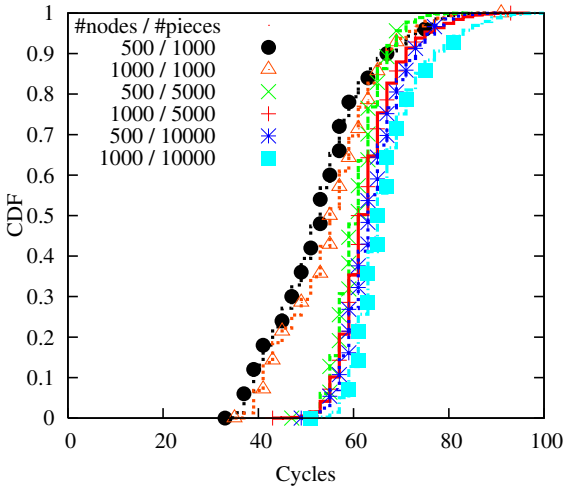


Fig. 2. CDF if the maximum delay (symmetric contact list; $k = 16$).

For the different scenarios, we perform 20 independent realizations and we compute the Empirical Cumulative Distribution Function (ECDF or simply CDF) by combining the realizations.

C. Fundamental Cycle-Based Properties

In this section we present the result for the cycle-based system. The purpose of this section is twofold: We validate our implementation against the results presented in [7] and we show other performance measures, in order to gain more insight into the operations.

For the validation, we consider the asymmetric overlay with various degree k. The results presented in [7] focused on the completion time and they showed that, for $k > 8$, it reaches a stable value approximately equal to $2C + 2\log N$. In Fig. 1(a) we show the full CDF of the completion time for different values of the contact list size (degree) k and for different values of number of pieces C. In order to be able to compare the results for different C, we show the completion time as the number cycles *after* the source has pushed the last piece. We observe that the completion time becomes stable for $k > 12$ (the upper group of curves; lower curves refer to $k = 8$), a value that is slightly higher than the one presented in [7]. Since there are not many details in the simulation methodology used in [7], we are not able to investigate the reason of these differences. In any case, the qualitative behavior —that the contact list can be kept small without jeopardizing the results— remains the same.

The performances in case of too small contact list size k are due do the difference in the number of incoming edges. The nodes responsible for the tail of the CDF are the ones that are poorly connected. In fact, in a random graph with mean degree $k = 8$ there is a non-negligible probability that nodes are only in one or two contact lists, or even in none, so that most pieces must be pulled actively, since the probability of being pushed is small. This problem is reduced if we increase the mean number of contacts. We found similar results for different overlay network sizes (not shown here for space constraints).

In case of symmetric overlay, each node has a constant number of contacts and the problem observed in the asymmetric case disappears. Figure 1(b) shows the CDF of the download time in case of symmetric contact list.

Comparing the CDF of the completion times in the different scenarios, we observe that the performances are almost equivalent, independently from the type of overlay (symmetric and asymmetric, provided that the degree is greater than 12), or from the number of pieces. This is valid for any performance measure we consider. For this reason hereinafter we will show only results where the contact list is symmetric and the degree equal to 16.

The evaluation in [7] focused on the completion time, since Interleave was proposed for file distribution. Nevertheless, the piece selection policy used by the scheme takes into account the order of creation of the pieces. For this reason it is interesting to study if the protocol can be used for distributing a stream. Figure 2 shows the CDF of $d^{\max}$, the maximum delay.

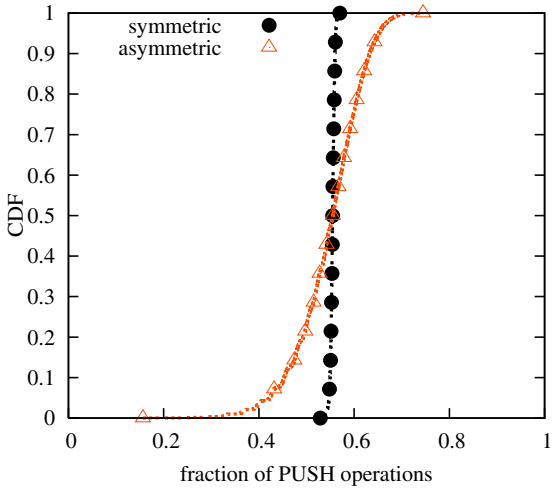


Fig. 3. CDF of the Push operations (symmetric contact list; $k = 16$; $N = 1000$; $C = 10^4$).

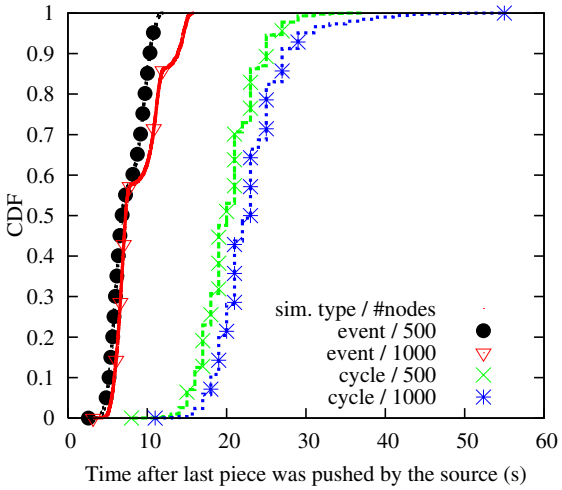


Fig. 4. CDF of the overall download time (symmetric contact list; $k = 16$; $C = 5 \cdot 10^3$; upload bandwidth: 256 kbit/s).

The maximum delay is slightly influenced by the number of pieces C, especially the maximum value of the CDF. This means that, by properly setting the initial delay, the streaming can be received at the application level without interruption.

An interesting future work will address the analysis of delay percentiles. Using Forward Error Correction or Network Coding techniques, it is sufficient to receive a percentage f of the information to reconstruct the whole flow. Analyzing the delay behavior as a function of the f-th percentile will give indications on the design of the multi-coding technique to be used, as well as on the amount of redundancy needed to support a streaming with a given maximum delay.

We notice that the maximum delay is expressed in number of cycles, thus finding efficient ways to reduce the piece size, and consequently the cycle duration, is an efficient way of supporting live streaming. Reducing the piece size to a single video frame (25 ms) will reduce the maximum delay to less than 2 s which is already a good value for streaming purposes.

Finally, in Fig. 3 we show the number of pieces received by push for different overlay types, asymmetric and symmetric. In both cases more pieces are received by push then by pull. In case of asymmetric overlay, the distribution has greater variance, since the overlay connectivity distribution has greater variance. The symmetric case shows instead almost perfect balance over all nodes, with slightly more pieces received via push.

D. Impact of the Realistic Model

While the cycle-based is interesting as a first step in the analysis, the realistic model is able to show if the scheme still maintains the good performances in a scenario with no synchronization (and possibly heterogeneous). We start comparing the completion time obtained from the cycle-based model with the results obtained with the realistic model. To this aim, in the cycle-based model we set the duration of the cycle equal to 0.5 s, while in the realistic model we set the upload bandwidth equal to 256 kbit/s.

Fig. 4 shows the CDF of the completion time with the two models: Not only the realistic model protocol is able to work without synchronization, but it obtains also better performances in term of download time. For the realistic model, we consider also different scenarios: Also in this case, the overlay connectivity type (asymmetric or symmetric) has no influence if the degree is greater than 12. Moreover, different number of pieces C and number of nodes N give similar results.

The factor that mainly influences the performance here is the upload bandwidth of the nodes (including the source). We considered the case with upload bandwidth close to r_{str}, the rate of the streaming (i.e., 128 kbit/s): In this case the system is unstable and it is not possible to reach convergence.

Fig. 5 shows the CDF of the maximum delay for two different upload bandwidths. The impact of number of nodes on the CDF, especially with an upload bandwidth equal to 256 kbit/s, is mainly due to the greater number of hops that pieces have to do to reach all the nodes. An interesting result is represented by the decrease of the maximum delay as the upload bandwidth increases. The initial delay for a streaming application that uses the Interleave protocol is only few pieces.

Analyzing the delay, it is interesting to investigate the scalability in terms of number of pieces and number of nodes. To this aim, we consider the maximum value of the CDF of the maximum delay, and we compare different scenarios. Fig. 6 shows how the delay varies as the number of pieces increases. Especially when the bandwidth is equal to twice the streaming rate, there is a non negligible increase in the maximum delay. This means that the scheme may not be able to sustain long streaming and different mechanisms that are able to maintain the delay bounded are necessary.

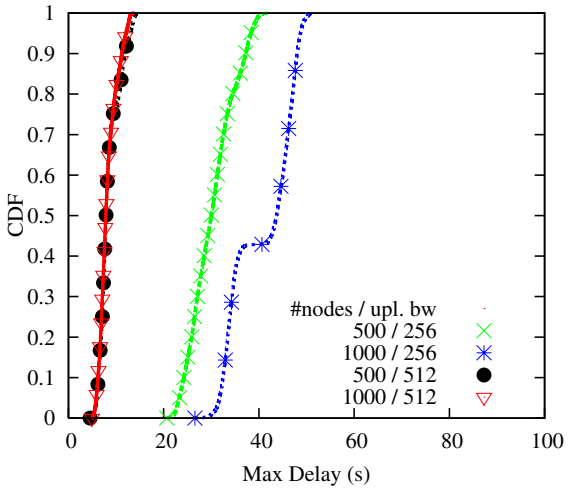


Fig. 5. CDF if the maximum delay (symmetric contact list; $k = 16$; $C = 5 \cdot 10^3$).

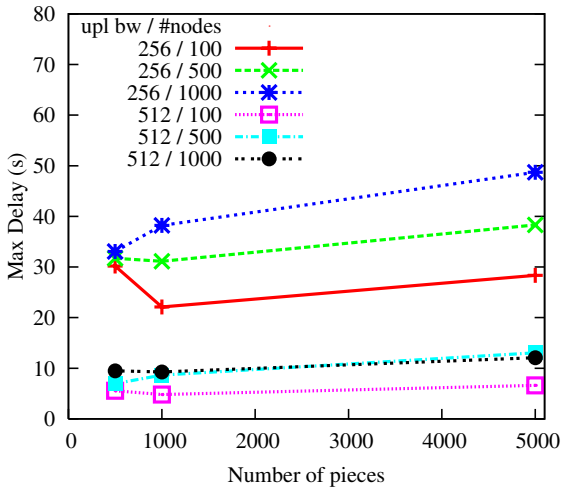


Fig. 6. Maximum value of the maximum delay (symmetric contact list; $k = 16$).

IV. Conclusions and Future Work

In this paper we considered a class of protocols suitable for supporting both file-based and stream-based communications in P2P overlay networks. We focused our attention on the information exchange process, considering protocols that alternate (of interleave) push and pull phases, i.e., a node alternates between helping some other node (pushing) to receive fresh information it just received itself, and filling its own information gaps by pulling information from neighboring nodes.

The advantage of the protocol we analyze is that it can work without making any assumption on the node behavior: The interaction between two nodes is limited to the exchange of a single piece, making it suitable to situations with high churn. We evaluate the performance of this basic scheme, considering that it provides a lower bound on the performance achievable by any system, since adding other mechanisms — like for instance spreading the information about pieces the nodes own— should normally improve the performance.

We think that these results will help to better understand content distribution systems and also make a contribution to the debate about the relative merits of basic, stateless schemes and status-based schemes.

Future research will focus on non-homogeneous systems, dynamic overlays with churn, and also multiple descriptions to sustain live-streaming over simple, entirely decentralized systems.

Acknowledgments

The authors wish to thank Ernst Biersack from Institut EURECOM for all the useful discussions and ideas about P2P streaming and distribution systems.

References

[1] Napster, `http://www.napster.com`
[2] T. Klingberg, and R. Manfredi, "The Gnutella Protocol Specification," `http://rfc-gnutella.sourceforge.net/src/rfc-0_6-draft.html`
[3] The Kazaa Network, `http://www.kazaa.com/`
[4] D. Carra, R. Lo Cigno, and E.W. Biersack, "Stochastic Graph Processes for Performance Evaluation of Content Delivery Applications in Overlay Networks," in *IEEE Transactions on Parallel and Distributed Systems*, Vol. 19, No. 2, pp. 247-261, Feb. 2008.
[5] D. Carra, R. Lo Cigno, and E.W. Biersack, "Graph Based Analysis of Mesh Overlay Streaming Systems," in *IEEE Journal on Selected Area in Communications*, Vol. 25, No. 9, pp. 1667-1677, Dec. 2007.
[6] E.W. Biersack, D. Carra, R. Lo Cigno, P. Rodriguez, and P. Felber, "Overlay Architectures for File Distribution: Fundamental Performance Analysis for Homogeneous and Heterogeneous Cases," in *Computer networks*, Volume 51, No. 3, pp. 901-917, February 2007.
[7] S. Sanghavi, B. Hajek, and L. Massoulie, "Gossiping with multiple messages," IEEE Transactions on Information Theory, Dec. 2007.
[8] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel, "ALMI: An Application Level Multicast Infrastructure," in *Proc. USITS*, Mar. 2001.
[9] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable Application Layer Multicast," in *Proc. SIGCOMM*, Aug. 2002.
[10] AD. A. Tran, K. A. Hua, and T. T. Do, "A Peer-to-Peer Architecture for Media Streaming," in *IEEE JSAC: Special Issue on Advances in Overlay Networks*, Vol.22, N.1, Jan. 2004.
[11] M. Castro, P. Druschel, A. M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: Highbandwidth Multicast in a Cooperative Environment," in *Proc. ACM SOSP*, Oct. 2003.
[12] V. Venkataraman, and P. Francis, "Chunkyspread: Multi-tree Unstructured Peer-To-Peer Multicast," in *Proc. IPTPS*, Feb. 2006.
[13] Y. Sung, M. Bishop, and S. Rao, "Enabling Contribution Awareness in an Overlay Broadcasting System," in *Proc. SIGCOMM*, Sept. 2006.
[14] Y.-H. Chu, S. G. Rao, and H. Zang, "A Case for End System Multicast," in *Proc. SIGMETRICS*, June 2000.
[15] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. Mohr, "Chainsaw: Eliminating Trees from Overlay Multicast," in *Proc. IPTPS*, Feb. 2005.
[16] X. Zhang, J. Liu, B. Li, and T. S. P. Yum, "DONet/CoolStreaming: A Data-driven Overlay Network for Live Media Streaming," in *Proc. INFOCOM*, Mar. 2005.
[17] F. Pianese, J. Keller, and E. W. Biersack, "PULSE, a Flexible P2P Live Streaming System," in *Proc. 9th IEEE Global Internet Symposium*, Apr. 2006.
[18] M. Zhang, Q. Zhang, and S. Yang, "Understanding the Power of Pull-based Streaming Protocol: Can We Do Better?" IEEE Journal on Selected Areas in Communications, Dec. 2007.
[19] T. Locher, R. Meier, S. Schmid, and R. Wattenhofer, "Push-to-Pull Peer-to-Peer Live Streaming," in *Proc. DISC*, Sept. 2007.
[20] PeerSim: A Peer-to-Peer Simulator, `http://peersim.sourceforge.net/`