Delay-Aware Push/Pull Protocols for Live Video Streaming in P2P Systems

Alessandro Russo, Renato Lo Cigno
DISI - University of Trento, Italy

***Abstract*—P2P video streaming is receiving enormous attention, and when video is involved, the efficient use of the network becomes a very important issue, specially if live applications are addressed. In this work we study properties of *Push/Pull protocols* for the exchange of video chunks in non-structured systems. Push/Pull protocols are a broad class of chunk exchange mechanisms where peers alternate phases where they actively send chunks to other peers, with phases where they seek for missing chunks from other peers. We focus on the properties of the protocol, trying to gain insight on the distributed exchange mechanism itself. Then, we explore how performances can be improved if peers, in selecting the peers to exchange information with, also consider network level parameters, namely the round trip delay.**

I. Introduction and System Assumptions

P2P distribution systems, where a single information source disseminate a content to a large number of peers, are receiving a lot of attention from both the scientific community and industry, with video services booming both in live streaming mode and on demand. Among these systems, those based on chunkization of the information and swarm-like distribution of the chunks over a non-structured overlay are the most successful, (see for instance commercial applications like PPlive, SopCast, UUsee, TVAnts, etc., or scientific literature [5], [3], [9]).

In this paper we introduce a new class of protocols with zero contention derived from [4], [13]. This class is based on Push and Pull phases executed by peers in order to distribute the content among participants. Push and Pull operations are asynchronous and active and passive node behaviors are independent, coupled only by the access to the networks. Signaling ensures contention-free chunk delivery with high bandwidth efficiency. Then we enhance the protocol with RTT (Round Trip Time) based delay-awareness to investigate how network-aware applications can improve not only network usage, but also their own performance.

The main differences between a streaming system and the dissemination of file-based contents are: i) streaming systems have real-time (hard or soft) requirements; ii) information generation in streaming systems is periodic; and iii) streaming systems do not require the dissemination of the entire content, but a certain percentage (e.g., 1% or 5%) of the information can be lost without jeopardizing the service.

II. Push/Pull Protocols

The Push/Pull protocols are inspired by [4]. However, the model analyzed there is a high level abstraction, with little resemblance to real-life protocols. We have defined an asynchronous, distributed protocol with proper state machines for both the transmitter and receiver side, defining signaling to avoid conflicts and chunks duplications, thus becoming friendly in resource usage. Signaling can be piggybacked on chunks when feasible, reducing overhead. Bandwidth overhead is in any case marginal, while the additional negotiation delay may ruin performance as discussed later. Contrary to [10] we do not aim at building implicit distribution trees, but at keeping a dynamic and flexible alternation of Push and Pull phases, which, as discussed in the sequel, achieve jointly the contrasting goals of diffusing fresh information to other peers and retrieve missing chunks locally. Early results on these protocols were discussed in [13].

In their most general form Push/Pull protocols work as described by the algorithm in Fig. 1. Each peer P_i is completely independent from the others in its decisions and there is no timing coordination, apart from the fact that chunks are timestamped from the source. This definition differs radically from the asymptotic analysis of [4], where global coordination is considered and the system works in *cycles* with all peers being synchronized either in Pull of Push state. The notation we use in this paper is reported in Table I. We usually refer to state with capital letter (e.g., Push) while the messages are with small letters (e.g., pull).

In general, a peer can follow any algorithm to select the chunk and peer for scheduling either the push or the pull (functions ChunkSelect and PeerSelect in Fig. 1). Then the peer will verify if the exchange is useful with a signaling message before proceeding to the actual chunk transfer itself. Finally, the protocol will decide the next Push/Pull state as a function of the current state and the result of the previous exchange.

Symbol	Description
$\mathcal{P}$, P_i	Set of peers; i-th peer
$\mathcal{N}_i$	Set of neighbors of peer i
$\mathcal{C}$; C_j	Chunks set composing the stream; j-th chunk
T_s	Chunk emission interval
α	Max number of active either uploads or downloads
ρ	Max number of passive either uploads or downloads
n_a	Max number of push or pull attempts
ω	Set of chunks to either push or pull

TABLE I
NOTATION AND SYMBOLS USED IN THE PAPER.

We are interested in protocols where peers do not exchange information about their state prior to scheduling so that

This work is supported by the European Commission through the NAPA-WINE Project (www.napa-wine.eu), ICT Call 1 FP7-ICT-2007-1, 1.5 Networked Media, grant No. 214412

the peers do not maintain any status relative to their neighbors, making the protocol inherently robust to dynamic neighborhood management. Specializing the algorithm in Fig. 1 to operate without knowledge about neighbors' state and with the alternation logic just described, the algorithm reported in Fig. 2 is obtained: each node actively contributes to the streaming by propagating fresh information via Push (*Latest* function); the Pull is devoted to retrieve locally missing chunks (*OldestMissing* function), thus it does not contribute to the streaming, but only to enhance local quality.

```
1: at every peer $P_i$
2: repeat
3:    if S == PUSH then
4:       $P_j$ = PeerSelect($\mathcal{N}_i$)
5:       $C_c$ = ChunkSelect($\mathcal{C}_i$)
6:       Transmit($P_j$,$C_c$)
7:       S = NextState(PUSH,Transmit)
8:    else if S == PULL then
9:       $P_j$ = PeerSelect($\mathcal{N}_i$)
10:      $C_c$ = ChunkSelect($\mathcal{C}_i$)
11:      Receive($P_j$,$C_c$)
12:      S = NextState(PULL,Receive)
13:   end if
14: until the stream is finished
```

Fig. 1. Generic algorithm for Push/Pull protocols at node $\mathcal{P}_i$,

The algorithm defines the *active* behavior of peers, when they propose an exchange. In particular, during Push, each node offers a set of chunks to some neighbors, actually pushing only to the first neighbor that accepts, while the others are refused explicitly; in Pull the node requests a set of chunks to its neighbors, accepting only the first positive reply, refusing the others. The protocol description is completed by the *passive* node's behavior, when a node is queried by other peers: each node may receive many offers of push, accepting only missed chunks and refusing the others, while it satisfies only one pull requests among all received. Clearly, a node can chose its own active state, so that it can decide to be either in Push or Pull state, but it cannot control the type of queries it receives, so that alternating between Push and Pull when queried it is not possible.

A window of ω chunks are offered or requested by peers during negotiation, giving some choice to the queried node, hopefully increasing the efficiency of the protocol. The node's state is changed deterministically from Push to Pull and vice versa upon success, or if the number of failed communication attempts is equal to n_a. Notice that a node can signal its offers and requests in parallel to multiple nodes to enhance its success probability. The higher the parallelism the higher the signaling overhead, but the higher the probability of success too. Considering that a signaling message is normally a single small packet, while a chunk can contain several hundreds kbytes of data, the signaling overhead remains small.

Albeit it is possible to execute Push and Pull phases in parallel (i.e., each node may send push and pull messages at the same time), we decided to maintain the alternation between the two states to limit the complexity of the system. This means that each node may be either in active Push or in Pull state, while the passive state depend on other nodes requests.

```
1: at every peer $P_i$
2: repeat
3:    $P_j$ = UniformRand($\mathcal{N}_i$)
4:    if S == PUSH then
5:       $\{C_c\}$ = Latest($\mathcal{C}_i$,$\omega$)
6:       Transmit($P_j$,$\{C_c\}$)
7:       if Tx-success OR Tx-failures==$n_a$ then
8:          S = PULL
9:       end if
10:   else if S == PULL then
11:      $\{C_c\}$ = OldestMissing($\mathcal{C}_i$,$\omega$)
12:      Receive($P_j$,$\{C_c\}$)
13:      if Rx-success OR Rx-failures==$n_a$ then
14:         S = PUSH
15:      end if
16:   end if
17: until the stream is finished
```

Fig. 2. Revised Push/Pull protocol with peer/chunks selection.

A. Delay-Aware Peer Selection

The algorithm in Fig. 2 selects one peer in the neighborhood at random. However this choice neglects network conditions, making the P2P distribution system network-agnostic: performance suffers and the network is often overloaded. There are many network properties and characteristics that can be exploited to improve the system behavior, however, most of them are very difficult to measure (e.g., available bandwidth between two peers). The RTT instead is very simple to measure accurately and is normally a very good indicator of how far nodes are and how much congested the network is between them. Thus trying to communicate with close-by (in terms of RTT) peers should improve performance and have a lighter impact on the network.

The main difference between the random and the delay oriented peer selection is that the second one picks those peers with lower RTT with higher probability, promoting chunks exchanges with closest neighbors. A simple way to do this is selecting peers in the neighborhood $\mathcal{N}_i$ based on a probability distribution function build as follows:

$$V_{i,j} = \frac{1}{RTT_{(i,j)}}\,, \quad i,j \in \mathcal{P} \tag{1}$$

$$SV_i = \sum_j V_{i,j}\,, \quad j \in \mathcal{N}_i \tag{2}$$

$$P_i(j) = V_{i,j}/SV_i \tag{3}$$

This choice reduces the average time $\overline{T}_c$ required to exchange each chunk between any peer, which in turn will significantly reduce the overall diffusion time, since the diffusion time is comprised between $\overline{T}_c|\mathcal{P}|$ for systems based on a distribution chain, and $\overline{T}_c\lceil\log_2|\mathcal{P}|\rceil$ for optimal systems that double the number of chunk replicas at every new transmission [6].

In some cases the delay oriented peer selection may suffer from repeated selection of the same neighbors, that reduces the correct spreading of information, building chains of peers instead of swarms. For this reason the peer selection is poisoned, so that the same peer can be selected again only after $2T_s$. We assume that a chunk can be transmitted within T_s, and we recall that nodes alternate Push and Pull, so the maximum time between two Push phases in the same node is $2T_s$, while on average multiple Push and Pull will alternate in this time. Without this threshold, a peer may push two or more consecutive chunks to the same peer, which may end up propagating only the last one via Push, while the others linger behind and will finally be retrieved via Pull, increasing their diffusion delay as a consequence. This poisoning guarantees a correct distribution of fresh content via Push.

III. Performance Metrics

We define the transfer delay of a chunk as

$$\delta_p(c) = T_p(c) - T_{out}(c)$$

where $T_p(c)$ is the time in which peer p receives chunk c, and $T_{out}(c)$ is the time the source generated chunk c. $\overline{\delta}_p(c)$ is the diffusion delay averaged over multiple realizations of the simulation. The following performance metrics are investigated:

a) CDF of the average chunks' transfer delay: The average diffusion delay for chunk c is

$$\overline{\delta}(c) = \frac{\sum_p \overline{\delta}_p(c)}{|\mathcal{P}|}, \quad c \in \mathcal{C}, p \in \mathcal{P} \tag{4}$$

Then, we compute the Cumulative Distribution Function (CDF) over all chunks.

b) CDF of the average transfer delay among peers: The average chunks' diffusion delay at peer p is

$$\overline{\delta}(p) = \frac{\sum_c \overline{\delta}_p(c)}{|\mathcal{C}|}, \quad p \in \mathcal{P}, c \in \mathcal{C} \tag{5}$$

We then compute the CDF over all peers.

c) Histogram of the chunks diffusion delay: It is the measured probability density function (pdf) of chunks' diffusion delay averaged over realizations. Here each chunk, at each peer, provides a point of the estimated pdf. The histogram is computed over $|\mathcal{C}| \times |\mathcal{P}|$ points. The bin size is 0.5 sec.

IV. Experimental Setup

We have implemented the push/pull protocols described in Sec. II in PeerSim (http://peersim.sourceforge.net/), which is a java-based P2P simulator.

The overlay topology graph $G(\mathcal{P}, \mathcal{E})$ is randomly built, in general having an n-regular random graph as a good model[1]. The topology is built selecting $|\mathcal{N}| \in \mathcal{P}$ neighbors at random for each node, with symmetrical connections: $e(i,j) \to e(j,i), i,j \in \mathcal{P}$ $(e(\cdot,\cdot) \in \mathcal{E}$ by construction). We use $|\mathcal{N}| = 16$; larger neighborhoods improve performance.

As parameter to tune the application to the network we consider the RTT for several reasons: i) The signaling phase is heavily affected by delay; ii) RTTs between peers are easily measured even at the application layer and are not much affected by correlations; iii) RTT delay is an indication of how far peers are in terms of hops, so that choosing closer peers will also reduce the global amount of network resources used. We assume that the size of control messages is negligible (a few tens of bytes), so that the message transfer delay on every connection $e(i,j)$ is roughly $1/2$ of the RTT. In the simulations we model the RTT of connections with a random variable extracted from a uniform distribution between $[\delta_{min} \div \delta_{max}]$ ms. RTTs from i to j and vice versa are the same.

We indicate the maximum number of parallel signaling messages sent by the node in active mode with α: α_{up} indicates messages that involve uploading (push offers), while α_{dw} is for messages involving downloading (pull requests). The passive node's behavior is controlled by ρ, which indicates how many of the messages received are positively answered: ρ_{up} refers to accepting requests that use the upload bandwidth (satisfying a pull), while ρ_{dw} concerns the download usage (receiving pushes).

The *source* streams at B_s Mbit/s, generating a new chunk every T_s s, and sends them to other peers. The source is a normal peer without additional resources. This is important because many evaluation studies assume that the source is somewhat special and has more upload bandwidth, which indeed modifies the performances non marginally. In order to guarantee the continuous emission of chunks, both negotiation and transmission of each chunk should be done within T_s. In accordance to these considerations, we always set peers upload bandwidth $B_p \geq T_s/(T_s - \delta_{max})$, in which is normalized to B_s. In our simulative study we use $T_s = 1$ s, thus for $\delta_{max} = 500$ the minimum value of B_p is 2.

PeerSim does not support actual packet transmission (which is one of the reason it allows simulating large overlays). We implemented a bandwidth management system based on priority sharing, which gives as much resources as possible to the first connection, then it tries to satisfy the second with the remaining part, and so on. This bandwidth mechanism is accurate enough to capture chunk transfer interaction and simple enough to maintain high simulation speed.

We consider a mildly popular stream with $|\mathcal{P}| = 1000$ peers and $|\mathcal{C}| = 4000$ chunks. We limit the accepted number of downloads to ten ($\rho_{dw} = 10$), allowing parallel downloads, and each node can satisfy at most one pull per time ($\rho_{up} = 1$). In Push each node proposes a window of size ω_{push} chunks to its neighbors, while in Pull it requests a set of ω_{pull} chunks. We usually set $\omega_{push} = \omega_{pull}$, referring to them with ω, as also $\alpha_{up} = \alpha_{dw}$, indicating their values with α. We limit the maximum number of attempts $n_a = 1$.

We investigate bandwidth homogeneous networks to simplify results interpretation. The system is at steady-state (peers in the overlay are stable). Peers try to retrieve all chunks of the stream, so that we can evaluate the tail behavior of the system; if an application may tolerate, say, 5% losses, it is enough to read pdf plots at 95^{th} percentile, as we do in Fig. 5.

[1]Indeed, the topologies considered in different works are not always n-regular topologies; however, most of the properties considered for mesh systems are not strictly dependent on the specific topology, whose main property is the randomness with a roughly constant connectivity degree, as confronted to specific, deterministic topologies such as trees or hypercubes.

V. Results

We discuss a set of results highlighting the role of the Push and Pull phases in the information delivery, and results showing the impact of RTT on the delay experienced by peers through the comparison of the Random (R) and Delay based (D) peer selection. Further details on this study can be found in [14].

A. The Role of Push and Pull

Fig. 3 highlights the different role of the Push and Pull states, and how they contribute to the pdf of chunks distribution. We can see that Push phase has lower delay and spreads the greater part of the chunks, while the Pull experiences the largest delay due to blind search for retrieving the missing chunks. We expect such a behavior because the Push is the active part of the distribution process, while Pull is a local recovery mechanism. The quantitative difference is however much larger than one can expect. The shape of the pulled chunks delay distribution shows lighter tails as B_p increases, since many more chunks are diffused via Push, so the number of pulls become smaller and the probability to find a chunk via pull becomes higher.

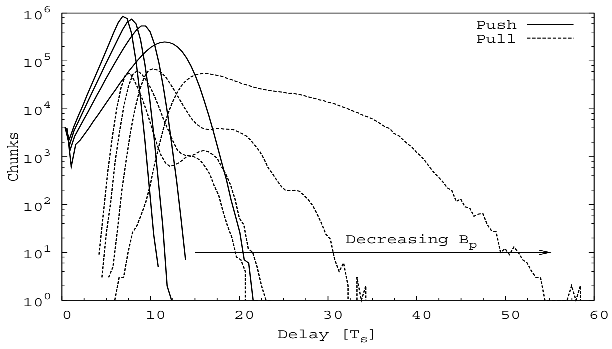

Fig. 3. Histogram of chunks' diffusion delay for Push and Pull mechanism, for $B_p = \{1.9, 2.4, 2.9, 3.4\}$, and RTT $[10 \div 250]$ ms.

B. Impact of Choice and Parallel Signaling

Fig. 4 shows the effect of varying the number of messages issued in each phase, and of providing a window of ω possible chunks during negotiation. First we focus on random peer selection, and then we describe the delay oriented one. We fixed $B_p = 1.9$, because in this situation the basic Push/Pull protocol ($\alpha = 1; \omega = 1$) does not provide satisfactory performance, as shown by the dashed line in Fig. 4. For $\alpha = 1$ and $\omega = 4$, offering multiple chunks leads to a clear reduction in diffusion delay, and the tail becomes smaller, as shown by the two-dotted curve. When we set $\alpha = 4$ and $\omega = 1$, the protocol achieves lower average diffusion delay, but the distribution tail remain important (short dashed line in Fig. 4), because some chunks are not properly diffused and remain rare. The combination of parallel signaling ($\alpha = 4$) and more choice among chunks negotiation ($\omega = 4$), reaches satisfactory results (dotted curve). The corresponding curve has a negligible tail above 12 s, and its mass is well concentrated around the mode at 9.55. Focusing now on the delay aware peer selection, always achieves better results, with negligible tails. In particular, with $\alpha = 1$ and $\omega = 1$ the delay aware selection limits the diffusion delay to roughly $14\,s$, while the corresponding curve with random selection has a tail extending beyond $20\,s$. When $\alpha = 1$ and $\omega = 4$, the curve becomes narrower and within $12\,s$, while for $\alpha = 4$ and $\omega = 1$, the curve shifts to the left, obtaining a meaningful diffusion delay reduction. Finally, for $\alpha = 4$ and $\omega = 4$, the combination of parallel signaling and the possibility of choosing among chunks, with the delay oriented peer selection, leads to lower delays than all the others, as shown by the solid line.

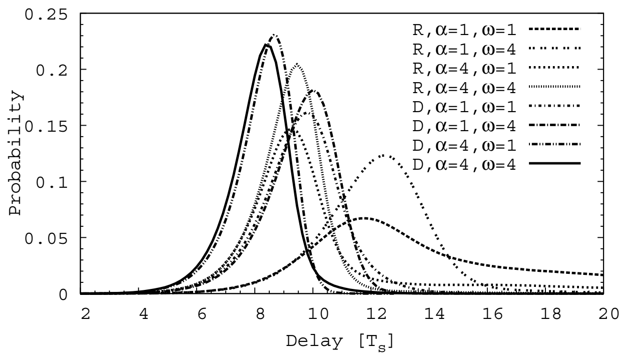

Fig. 4. Histogram of chunks' diffusion delay among all peers for different values of ω, α: B_p=1.9, and RTT $[10 \div 250]$ ms.

C. Impact of Upload Bandwidth

We continue our analysis allowing both parallel signaling and allowing choice during chunks negotiation, i.e., with $\alpha = 4$ and $\omega = 4$. Fig. 5 presents the 95^{th}-percentile of chunks' diffusion delay, for different uploads configurations, and RTT distribution $[10 \div 250]$ and $[10 \div 500]$ ms. Again, the delay aware peer selection shows that our intuition is right, achieving an average diffusion delay lower than the random peer selection one, especially with small upload bandwidth. Increasing the maximum RTT, the delay aware selection shows a larger gain even when bandwidth resources are very high. Although we double the RTT delay, the delay aware selection does not reflect a proportional increase in chunks' diffusion delay for the same upload bandwidth: the difference remains around 1.5 times.

In Fig. 6 we focus our attention on RTT between $[10 \div 500]$ ms, showing its impact for different upload bandwidth. In this case the RTT heavily impacts chunks' diffusion delay, and increasing the upload bandwidth the RTT becomes the main part of chunks' diffusion delay. For this reason neighbors selection should be done carefully, reducing the impact of the RTT and its propagation in chunks' distribution. The curves become narrower as B_p increases, for both random and delay oriented selection. However, the delay oriented approach always achieves lower delay than the random one, reducing the average chunks' diffusion delay experienced by peers, giving useful information for computing buffering time and timeout for chunks to be pulled.

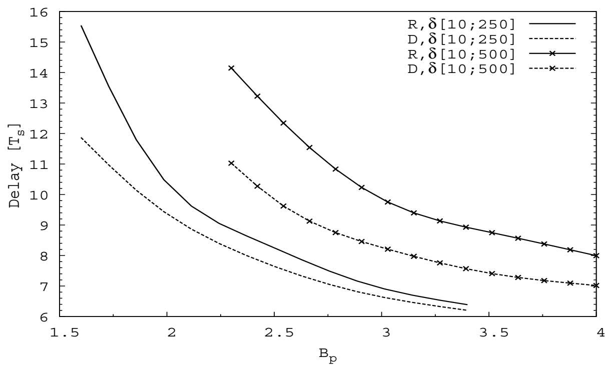

Fig. 5. 95^{th}-percentile chunks' diffusion delay for different B_p and RTT.

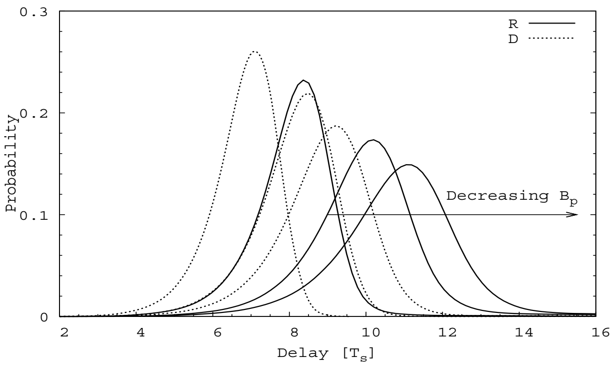

Fig. 6. Histogram of chunks' diffusion delay for $B_p = \{2.3, 2.5, 3.0\}$, and RTT $[10 \div 500]$ ms.

Fig. 7 shows the comparison between the delay oriented and the random peer selection, comparing the cdf of the average delay experienced by peers during the streaming. The delay experienced by peers are quite stable, and the curves rise quickly, showing that the inter-arrival chunks' delay in each peer is small and steady. Moreover, the delay oriented approach confirms that the choosing of closest neighbors leads to low delay, without big dispersions. Note that with just 2.3 time B_s, with RTT $[10 \div 500]$ ms, the delay oriented is within $12\,s$, while the corresponding random is around $16\,s$, and the average distance between them is about 30%.

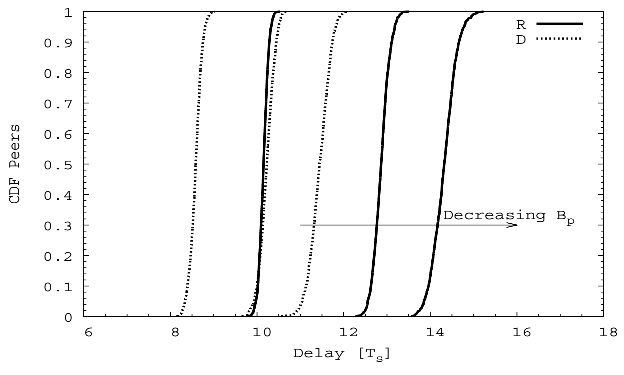

Fig. 7. CDF of the $\overline{\delta}(p)$, $B_p = \{2.3, 2.5, 3.0\}$, and RTT $[10 \div 500]$ ms.

VI. Conclusions

This paper discussed Push/Pull protocols which alternate phases where information is pushed to neighbors with phases where information is pulled from neighbors. A key result of this paper is that RTT is a very important parameter for delay sensitive systems, so peer selection algorithms which promote communication among closest neighbors are more suited, even with limited resources. We stress that the proposed system is bandwidth-optimal in that the only useful information (and signaling) is distributed, avoiding the waste of many P2PTV systems. Moreover, the ability to choose during chunks negotiation, combined with parallel signaling, lead to lower diffusion delay. The overall insight gained on the Push/Pull system shows that with resources which are no larger than two times the stream rate, the streaming is sustainable and efficient (the protocol ensures small overhead), a result that with protocols exploiting only push or pull mechanism is achieved only at the price of state exchange between peers.

Further research include exploring churn impact, heterogeneity of nodes and different overlay construction mechanisms, as well as better network models, and more efficient implementations of the Push/Pull protocol, e.g., by allowing signaling to be in parallel with data transmission.

References

[1] D. Qiu and R. Srikant, "Modeling and Performance Analysis of BitTorrent-Like Peer-to-Peer Networks", in *Proc. ACM SIGCOMM 2004*, Portland, OR, Sept. 2004.

[2] D. Stutzbach, D. Zappala, and R. Rejaie, "The Scalability of Swarming Peer-to-Peer Content Delivery", in *Proc. Networking 2005*, Waterloo, Ontario, Canada, May 2005.

[3] 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.

[4] S. Sanghavi, B. Hajek, and L. Massoulie, "Gossiping with multiple messages," *IEEE Trans. on Information Theory*, Vol. 53, No. 12, pp. 4640–4654, Dec. 2007.

[5] AD. A. Tran, K. A. Hua, and T. T. Do, "A Peer-to-Peer Architecture for Media Streaming", *IEEE JSAC: Special Issue on Advances in Overlay Networks*, Vol.22, N.1, Jan. 2004.

[6] L. Abeni, C. Kiraly, and R. Lo Cigno, "On the Optimal Scheduling of Streaming Applications in Unstructured Meshes", in *Proc. Networking 2009*, Aachen, Germany, May 2009.

[7] 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.

[8] T. Locher, R. Meier, S. Schmid, and R. Wattenhofer, "Push-to-Pull Peer-to-Peer Live Streaming", in *Proc. 21-st DISC*, Sept. 24–26, Lemesou, Cyprus, LNCS 4731, Springer, 2007.

[9] N. Magharei and R. Rejaie, "Prime: Peer-to-Peer receiver-driven mesh-based streaming, In *Proc. INFOCOM 2007*, May 6–12, 2007.

[10] M. Zhang, Q. Zhang, L. Sun and S. Yang, "Understanding the Power of Pull-Based Streaming Protocol: Can We Do Better?", *IEEE JSAC*, Vol. 25, No 9, pp. 1678–1694, Dec. 2007.

[11] T. Bonald, L. Massoulié, F. Mathieu, D. Perino, and A. Twigg, "Epidemic live streaming: optimal performance trade-offs", in *Proc. SIGMETRICS 2008*, Annapolis, Maryland, USA, pp. 325–336, June 2-6, 2008.

[12] A.P. Couto da Silva, E. Leonardi, M. Mellia, and M. Meo, "A Bandwidth-Aware Scheduling Strategy for P2P-TV Systems", in *Proc. IEEE P2P'08*, Aachen, DE, September 2008.

[13] R. Lo Cigno, A. Russo and D. Carra, "On some fundamental properties of P2P push/pull protocols", in *Proc. HUT-ICCE'08*, HoiAn, Vietnam, June 2008.

[14] R. Lo Cigno and A. Russo, "Delay-Awareness in Push/Pull Streaming Protocols", University of Trento, Tech. Rep. TR-DISI-10-012, 2010. Available at http://disi.unitn.it/locigno/preprints/TR-DISI-10-012.pdf