Scheduling in P2P Streaming: From Algorithms to Protocols

Luca Abeni and Alberto Montresor*

DISI - University of Trento, Trento (IT)
{luca.abeni, alberto.montresor}@unitn.it

Abstract. Chunk and peer scheduling is among the main driver of performance in P2P streaming systems. While previous work has analytically proved that optimal scheduling algorithms exist, such strategies are based on a large number of strong assumptions about the knowledge that a single peer has of the rest of the system. This short paper presents a protocol for turning these theoretical results into practical ones, by taking into account practical aspects like the diffusion time of signaling messages and a partial knowledge of the participating peers.

1 Introduction

Peer-to-peer streaming systems are becoming increasingly popular [1,2,3] as a way to overcome the scalability limitations of traditional streaming technologies based on a client/server paradigm. In particular, there is an increasing interest in *unstructured* P2P streaming solutions, which distribute a media stream by dividing it in *chunks* that are disseminated by the various peers contributing to the stream diffusion. This approach allows to better exploit the upload bandwidth of all the peers which actively participate in the dissemination, and to reduce the overhead required for maintaining the overlay structure in case of churns. Moreover, unstructured systems are more tolerant to peer failures [4].

Traditionally, unstructured P2P streaming systems tend to distribute the chunks based on *random* decisions, generating a lot of useless network traffic (due to duplicated chunks) and delaying the chunk diffusion. These effects can be avoided by basing the chunks distribution on smarter decisions (this is the scheduling problem), so that every peer can receive the various chunks at the proper time and chunks duplications (the same chunk sent to the same peer multiple times) are reduced or completely eliminated. There has been a good amount of research on scheduling in P2P streaming systems, and some scheduling algorithms providing good mathematical properties have been developed [5,6,7]. Some of such algorithms (namely, Rp/Lb, Rp/LUc, MDp/LUc, LUc/ELp, and Dl/ELp) have been formally analysed, and various optimality properties have

* This work is supported by the European Commission through the NAPA-WINE Project (Network-Aware P2P-TV Application over Wise Network – www.napa-wine.eu), ICT Call 1 FP7-ICT-2007-1, 1.5 Networked Media, grant No. 214412.

T. Spyropoulos and K.A. Hummel (Eds.): IWSOS 2009, LNCS 5918, pp. 201–206, 2009.

been proved (see the papers cited above for details on the algorithms and on their properties).

An unstructured P2P streaming system is modelled as a set of nodes in which a special one, called the *source*, generates chunks at a fixed rate from an encoded media stream. Chunks are distributed to the peers present in the system through a *push* strategy: each peer contributing to the streaming selects a chunk to be sent and the target peer, and sends (pushes) the chunk to the target. The goal is to select the chunk to be sent and the target peer so that duplicated chunks are reduced and the streaming performance is improved. These two decisions are taken by two *schedulers*, called *chunk* and *peer scheduler*. The peer scheduler can select a target peer from a set of peers named *neighbourhood* (basically, each peer knows the existence of a limited subset of peers).

The works cited above focused on the scheduling algorithms, assuming that each scheduler has some knowledge of the peer's neighbourhood (and knows the chunks that have already been received by all the neighbours). Since such algorithms are often based on a set of assumptions regarding the overlay management or the diffusion of the information about the chunks that have been received by each peer, the creation and maintenance of the neighbourhood (overlay management) and the signalling mechanisms between different peers (chunk buffer state management) have not been considered. Hence, the theoretical properties proved in the original papers have to be verified in more realistic situations.

This paper presents the design of PushStream, a protocol that can be used to implement various peers and chunks scheduling algorithms, and evaluates the performance of such protocol through a set of simulations which take in account the overlay construction, the signalling delay, and churn.

2 The PushStream Protocol

As explained, to implement an effective scheduler the scheduling algorithm must be complemented with a protocol which takes care of the chunk buffer state management and overlay management issues.

Some overlay management protocols which are well known in the P2P community (such as Newscast [4]) can be used to construct the neighbourhoods, and chunk buffer state management can be performed by sending signalling messages when a peer receives a chunk. However, the overlay management algorithms generally create unidirectional links (that is, if peer P_i is in P_j's neighbourhood, it is not guaranteed that P_j is in P_i's neighbourhood), while the solution mentioned above requires overlays based on bidirectional graphs (because when a peer P_i receives a chunk, it should notify all the peers that can potentially send chunks to P_i - that is, all the peers that have P_i in their neighbourhoods). In the following, this problem will be referred as the *bidirectional neighbourhood* problem.

Another problem is caused by the one-way-delay: if peer P_i receives a chunk C_k at time t and sends a message to peer P_j to notify it about the chunk reception, P_j will receive such notification only at time $t' = t + \delta$, where δ is

the one-way-delay. Hence, if P_j's scheduler runs between t and t' then it might decide to send again C_k to P_i. In the following, this problem will be referred as the *delayed notification* problem.

The protocol proposed in this paper, PushStream, addresses the bidirectional neighbourhood problem by dynamically constructing an *input neighbourhood*, which is used to send notification messages: when peer P_i receives a chunk from peer P_j, P_j is added to P_i's input neighbourhood, and a notification message is sent to all the peers in the input neighbourhood (all but P_j, clearly).

The delayed notification problem is particularly dangerous when using "more deterministic" schedulers such as ELp [7] which try to take optimal decisions to reduce the chunks diffusion times as much as possible. In fact, such algorithms increase the probability to have multiple peers sending the same chunk to the same target simultaneously (because of the "deterministic" behaviour of the scheduler, many peers will take the same decision as they are unaware of what the other peers are doing). This problem can be addressed by exploiting some topological properties to reduce the collisions probability: basically, the scheduler does not select the target from the whole neighbourhood provided by the overlay management protocol, but from an *output neighbourhood*, which is a subset of such neighbourhood. In particular, the output neighbourhood of P_i is composed by P_i's neighbours based on some desirable property (in this paper, the minimum topological distance from P_i has been used).

Hence, each peer simultaneously runs an overlay management protocol and PushStream, which is a cyclic protocol (with time cycle T) and works as follows:

- When a chunk C_k is received from a peer P_j
  - Add C_k to the chunk buffer
  - Add P_j to the input neighbourhood N^{in}, if not already present
  - Otherwise, reset P_j's timeout to the maximum value T^{out}
  - For each peer $P_h \in N^{in}$, send a notification about C_k reception to P_h, and decrease P_h's timeout
  - If the timeout of P_h is 0, remove P_h from N^{in}
- When a notification message is received from peer P_j
  - Update the information about the chunks received by P_j
- At every cycle (with period T)
  - Invoke the scheduler to select a target peer in the output neighbourhood N^{out} and a chunk to be sent. The scheduler uses the information provided by the previous notification messages
  - Send the selected chunk to the target
- Every K cycles
  - Update N^{out} by using the peers known by the gossipping protocol having the smallest distance from the current peer.

It is worth noting that PushStream does not require any notion or global time, nor it assumes any kind of special synchronisation between the peers.

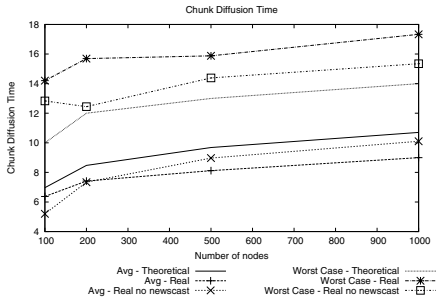

Fig. 1. Chunk diffusion time in the ideal case and using PushStream

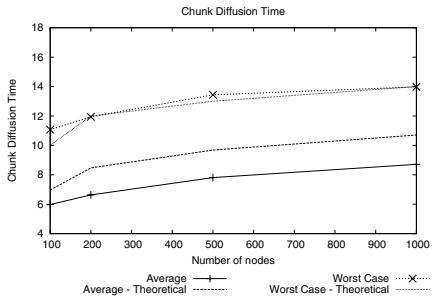

Fig. 2. Chunk diffusion time in realistic situations using PushStream

3 Preliminary Results

The PushStream protocol has been implemented in the Peersim[1] P2P simulator, which also emulates the chunk transmission time and the signalling delay, and its performance have been evaluated and compared with the "theoretical results".

In these PushStream tests, Dl/ELp [7] has been selected as a scheduling algorithm (because of its optimality properties), and the Newscast algorithm has been used for overlay management. The simulations have been run assuming a chunk size of $1s$ (each chunk contains 1 second of encoded media), an output neighbourhood size equal to $\log_2(N)$, where N is the number of peers, and a Newscast neighbourhood size equal to $3\log_2(N)$.

Correctness of the Implementation: The correctness of the implementation has been verified by simulating an "almost ideal situation" (all the chunk transmission times and the signalling delays are set to the minimum possible value, which is $1ms$) and comparing the results with the theoretical results from [7][2]. The streaming performance have been evaluated by considering the chunk diffusion time (the time needed by a chunk to be diffused to all the peers): in particular, Figure 1 displays the worst case and average values for the chunk diffusion times. The average values obtained by using PushStream are smaller than theoretical ones because each peer receives about 93% of the chunks, and the statistics are computed on the received chunks only (in the theoretical case, no chunk is lost). Such lost chunks are mainly due to Newscast dynamically changing the overlay, and to the initial startup time needed to setup the input neighbourhood (during this time, a lot of duplicated chunks are received). To verify this, the simulations have been repeated disabling the periodic update of

[1] http://peersim.sf.net

[2] Such reference values have been obtained by simulating the ideal system (in which every peer knows the exact state of all its neighbours, the overlay is static, and the neighbourhood size is assumed to be equal to $3\log_2(N)$).

the output neighbourhood (in this way, PushStream ends up using a static overlay, and Newscast does not affect the streaming performance); as a result, each pears receives more than 98.5% of the chunks and the worst case diffusion times are a little bit larger than the theoretical bound as shown in Figure 1. In any case, the measured values are quite close to the theoretical ones, showing that the protocol is correctly implemented.

Realistic Setup: In the previous simulations, the size T of the periodic cycle of each peer has been set to $1s$ because the Dl/ELp algorithm is known to be able to diffuse a media stream when all the peers forward the stream at its bitrate (hence, since each chunk is $1s$ large the system can work if each peer outputs one chunk per second). Since in a real system there will be some duplicate chunks, the output bitrate requested to each peer is larger than 1; hence, in the next set of simulations the cycle size of each peer but the source will be set to $T = 1/1.2$.

In a next batch of simulations, PushStream has been simulated in more realistic situations, assuming that a chunk needs a time randomly distributed in $[100ms, 300ms]$ to be transmitted, and the one-way-delay for the signalling messages is randomly distributed in $[50ms, 200ms]$. The results are reported in Figure 2, which again displays the average and the worst case diffusion times, compared to the theoretical values from [7]. By looking at the figure, it is possible to see that the algorithm performance are still near to the theoretical bounds (again, the average diffusion time is smaller than the theoretical one because of the lost packets, and because $T < 1$). In these simulations, the average percentage of chunks not received by a peer is always less than 6%.

Overhead Measurements: A potential problem with PushStream is that the size of the input neighbourhood can grow too much, forcing a peer to send a lot of signalling messages each time that it receives a new chunk. Hence, the size of the input neighbourhood for the previous simulations has been measured, and it turned out that the maximum value is 20 (forced by the timeout used to remove peers from the input neighbourhood) and the average value is less than 16. Hence, the overhead introduced by the signalling protocol is not too high.

Dynamic Overlay Management: To evaluate the effect of the dynamic overlay construction, the simulations have been repeated disabling the periodic update of the output neighbourhood. As a result, it has been observed that the simulation results became heavily affected by the overlay topology: in some runs, the chunk diffusion delays and the fraction of lost chunks became very similar to the theoretical expectations, while in other runs the performance became poor. When, instead, Newscast is used to dynamically update the output neighbourhood, the results become very consistent from run to run.

Additional Results: To understand the effect of the output bitrate on the protocol performance, the number of peers has been fixed to $N = 500$ (so, the neighbourhood size is 27 and the output neighbourhood size is 9), and the cycle size has been set to $T = 1.0/(1.0 + \rho)$, (where ρ is called *surplus bandwidth*).

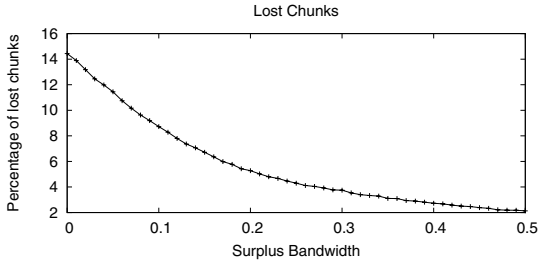

Fig. 3. Chunk loss for 500 peers as a function of ρ, where $T = 1.0/(1.0 + \rho)$

The results obtained increasing ρ from 0 to 0.5 are shown in Figure 3: note that if peers can forward the chunks at a bitrate that is 130% of the stream bitrate, then the average amount of lost chunks is less than 4%.

Finally, some simulations have been run to verify how the system copes with peers that leave the system without any kind of notification. The experiments showed that the dynamic update of the input and output neighbourhoods allow to tolerate such "leaving peers" without affecting the performance too much: for example, if 1% of the peers leave the system every $500s$, in a system composed by 1000 peers each peer receives about 92% of the chunks (without churn, around 95% of the chunks were received).

References

1. Hefeeda, M., Habib, A., Xu, D., Bhargava, B., Botev, B.: Collectcast: A peer-to-peer service for media streaming. ACM Multimedia 2003 11, 68–81 (2003)
2. Liu, Y.: On the minimum delay peer-to-peer video streaming: how realtime can it be? In: MULTIMEDIA 2007: Proceedings of the 15th international conference on Multimedia, Augsburg, Germany, September 2007, pp. 127–136. ACM Press, New York (2007)
3. Couto da Silva, A., Leonardi, E., Mellia, M., Meo, M.: A bandwidth-aware scheduling strategy for p2p-tv systems. In: Proceedings of the 8th International Conference on Peer-to-Peer Computing 2008 (P2P 2008), Aachen (September 2008)
4. Jelasity, M., Voulgaris, S., Guerraoui, R., Kermarrec, A.-M., Steen, M.v.: Gossip-based peer sampling. ACM Trans. Comput. Syst. 25(3), 8 (2007)
5. Massoulie, L., Twigg, A., Gkantsidis, C., Rodriguez, P.: Randomized decentralized broadcasting algorithms. In: 26th IEEE International Conference on Computer Communications (INFOCOM 2007) (May 2007)
6. Bonald, T., Massoulié, L., Mathieu, F., Perino, D., Twigg, A.: Epidemic live streaming: optimal performance trade-offs. In: Liu, Z., Misra, V., Shenoy, P.J. (eds.) SIGMETRICS, Annapolis, Maryland, USA, June 2008, pp. 325–336. ACM Press, New York (2008)
7. Abeni, L., Kiraly, C., Cigno, R.L.: On the optimal scheduling of streaming applications in unstructured meshes. In: Networking 2009, Aachen, DE, May 2009. Springer, Heidelberg (2009)