PULSE, a Flexible P2P Live Streaming System

Fabio Pianese
France Telecom - Division R&D
Issy-les-Moulineaux, France
fabio.pianese@rd.francetelecom.com

Joaquín Keller
France Telecom - Division R&D
Issy-les-Moulineaux, France
joaquin.keller@rd.francetelecom.com

Ernst W. Biersack
Institut Eurecom
Sophia-Antipolis, France
erbi@eurecom.fr

Abstract— **With the widespread availability of inexpensive broadband Internet connections for home-users, a large number of bandwidth-intensive applications previously not feasible have now become practical. This is the case for multimedia live streaming, for which end-user's dial-up/ISDN modem connections once were the bottleneck. The bottleneck is now mostly found on the server side: the bandwidth required for serving many clients at once is large and thus very costly to the broadcasting entity. Peer-to-peer systems for on-demand and live streaming have proved to be an encouraging solution, since they can shift the burden of content distribution from the server to the users of the network. In this work we introduce PULSE, a P2P system for live streaming whose main goals are flexibility, scalability, and robustness. We present the fundamental concepts of PULSE along with its intended global behavior and describe in detail the main algorithms running on its nodes.**

I. Introduction

The distribution of multimedia content over a network can be performed either *on demand* or *live*. On demand distribution is suitable for media files that can be reproduced at any moment, with loose timing constraints such as some maximum delay between user request and start of playback. Live streaming involves a continuous flow of information that is (supposedly) being generated *at that instant* and is thus bound to be immediately consumed; media streams, given their ephemeral nature, are by definition quite sensitive to playback delay. Our work focuses on the problem of live media distribution to large user populations with heterogeneous network access capabilities, as found in today's Internet.

Native group communication infrastructures would be an efficient solution to this problem. Unfortunately, IP multicast suffers from incomplete deployment in the core of the network and lacks support by most commercial ISPs. Therefore distributing data from one source to multiple destinations requires an applicative solution.

The traditional client-server model is suitable for live streaming. The server must directly provide the stream to all of its clients: it needs sufficient computational resources and, most importantly, very large network pipes. The system must be dimensioned for a maximum number of simultaneous users, which determines the required upstream bandwidth. Sudden peaks of activity (flash crowds, slashdot effect, etc.) are common phenomena that may lead to disruption or denial of service. For this reason, servers are often over-provisioned with capacity and resources that won't be used most of the time. The costs incurred by the media provider can thus grow very high.

Peer-to-Peer networks have emerged in the last few years as a promising approach to solve these cost issues [1]. A common aspect of these systems is the shift of the distribution load and serving responsibility from the streaming source to all the entities in the P2P network. The basic building blocks, called *nodes* or *peers*, are no longer passive receivers of the data but can act both as clients and servers at the same time: stream data are simultaneously received, played, and passed on to fellow peers. The existing P2P live streaming systems present different architectures. They can be roughly classified into three main families:

a) Structured: in these systems, nodes are organized following a hierarchical tree structure to form an application-layer overlay network. Each node receives the whole data stream from its *parent* and transmits it to its *children* (if any). Differences between systems of this family concern the way nodes are organized and the algorithms used to build and repair the system. Some of the systems in this family are SpreadIt [3], PeerCast [4], ESM [5], NICE [6], ZigZag [7].

b) Unstructured: in these systems, the associations between the nodes are driven by the data they have previously received: the stream is broken by the source into chunks that are then made available to the peers. Nodes independently request and download the chunks they need to complete the stream. The concept of overlay network is thus not entirely appropriate to describe such a dynamic data trading mesh of connections established by the nodes. Systems like DONet/CoolStreaming [8] belong to this family.

c) Other: these systems do not belong strictly to one of the previous families. They range from hybrid architectures (structured control overlay superposed to unstructured data trading mesh) such as Bullet [9], to systems that combine coding techniques with multiple structured overlays (stream split into multiple flows that are sent through disjunct trees) such as Splitstream [2].

A common assumption made by all these systems is that all the peers can (and want to) cooperate to the replication of the stream. NICE and ZigZag require for instance that most peers have a guaranteed minimum outbound bandwidth, i.e. an integer multiple of the stream bandwidth, while Bullet and Splitstream suppose that all nodes will take an active role and contribute at least as much as they are receiving. In DONet/CoolStreaming, the observed link capacity (both incoming and outgoing) is used to improve the mesh quality over time.

To the best of our knowledge, no existing system tries to reward user participation and to discourage peers from contributing an insufficient amount of resources. Current approaches ignore the fact that peers can not meet the minimum outgoing bandwidth for technical reasons (asymmetric access technology such as ADSL, firewall issues) or because of selfish user behavior (tampering with the peer's algorithms, artificial data rate limitation, etc.). With PULSE, we attempt to address this issue by introducing various forms of incentives to cooperation in an unstructured, data-driven P2P live streaming system. These incentives, which are inspired to the 'altruistic tit-for-tat' strategy used by BitTorrent [13] in the field of bulk data distribution, were specifically adapted to take into account the dynamic nature of live data streams.

In the next section, we describe the PULSE system, both at the global and at the peer level. In Section 3 we present the main algorithms executed by a PULSE peer. Finally, we draw our conclusions in Section 4.

II. PULSE: System Description

We argue that placing nodes in the system according to their current trading performances is an important way to improve the data replication efficiency, both in terms of average reception delay and data loss rate. For instance, if nodes whose resources are scarce were systematically served recent data, they could slow down or even disrupt the distribution process. Intuitively, resource-rich nodes located near the source are able to serve a larger number of neighbors with more recent data. This placement can reduce the overall average reception delay for the whole network, given a fixed serving capacity. Moreover, nodes should also be able to freely roam in the system and to react to global membership changes and local bandwidth capacity variations over time.

PULSE is a P2P unstructured live streaming system built around these principles. It is designed to operate in scenarios where the nodes can have heterogeneous and variable bandwidth resources. The approach used by PULSE is data-driven and receiver-based. Peers implement data exchange policies that enforce and reward node cooperation. These incentives to resource contribution allow a gradual improvement in the quality of the data trading mesh and help reduce the average stream reception delay.

A. Network Overview

All PULSE nodes are functionally identical. They are free to exchange control information and data from the stream. Associations between nodes are the result of independent decisions, leading to pairwise negotiations and data exchanges. The connections, which are independently established by the peers, form a system-wide mesh of data trading/control links. This mesh is difficult to describe analytically, since it is not built following global criteria and since it depends on the current and past state of the data exchange in the system. Data exchange performances are in turn determined by many parameters. The most important ones are the topological characteristics of the underlying network, the resources available at each peer, and the chunk distribution/retrieval algorithms executed by the nodes.

B. About the Media Stream

The stream is a continuous flow of media data encoded at the streaming source. In this paper we assume that the stream bit-rate (SBR) is constant, even if this is not required. The source then splits the encoded stream into a series of *chunks*. At this stage, the source may apply to the data fixed-rate error correction codes, such as FEC [10], or other forms of redundant encoding, such as MDC [11], to achieve better resilience to chunk loss.

Chunks are numbered and marked with their original encoding time at the source (we call this time reference the *media clock*) to allow peers to correctly rebuild the initial stream and estimate their own play-out delay. The source may also want to protect data integrity by adding a digital signature to each chunk. This requires every PULSE node receiving the stream to know the source's public key and verify the signature on each chunk. The integrity mechanism is mainly useful when there is the risk of data pollution by malicious peers.

Chunks are then made available to all the PULSE nodes at a fixed rate. In our case, where SBR is constant, the chunk size is also constant, otherwise it would fluctuate around an average size.

C. Lag Reference System

In Figure 1 we illustrate the fundamental concepts and variables used throughout this paper. The horizontal axis represents the *lag*, which is defined as the age of a chunk with respect to the current media clock. The lag value of a given chunk grows linearly over time, as new data are encoded at the source and present data become older.

We chose this 'differential' reference system because it will ease the representation of the buffer dynamics. For instance, since the play-out rate is constant, the chunk a node should be playing at any given moment is described by a fixed lag value, which we will call T_V. This notation also allows us to define the range of chunks a node is both interested in receiving and capable to provide at some point in time by two values: the average lag of the chunks the node is requesting, $T_{B_{avg}}$, and the lag of the chunk a node is going to discard from its buffer, T_D. The value of $T_{B_{avg}}$ is the average of recent $T_{B_{inst}}$ values sampled over a fixed time frame. From now on, we will call the interval $[T_{B_{avg}}, T_D]$ the *buffer delay range* of a node.

This notation is mainly useful for the phase of peer discovery, when it is important to find nodes that are able to provide useful chunks. We can imagine that, when the system is in a steady state, nodes tend to settle on constant average reception delays. In this situation, to discover a potential partner, it is sufficient to compare at any time the nodes' buffer delay ranges. This can eliminate the need of continuously sending and requesting updated buffer information on a chunk-by-chunk basis. On the other hand, when the system is not in steady state, nodes' buffer delay ranges can fluctuate. However, the information on the buffer delay range is still

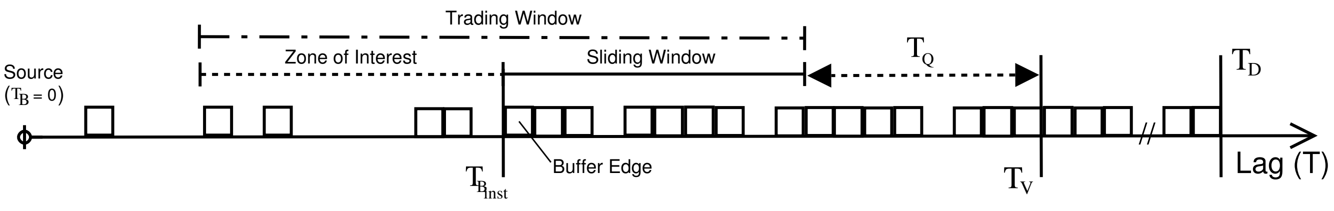


Fig. 1. A PULSE node's data buffer

much less volatile than the information on single chunks or chunk ranges: in normal operating conditions (i.e. while most nodes manage to retrieve a sufficient number of chunks on a regular basis), a node's reception delay will typically change quite slowly over time. It is thus still possible to use the buffer delay ranges, within a reasonable time frame from their computation, as an approximate and concise representation of the remote nodes' current buffer content.

D. Peer's Structure

A PULSE peer is an application that interfaces with the network to steadily retrieve data chunks and control messages. Its goal is to reconstruct the original stream of media data and to pass it on to the software player. Its main components are 1) the *data buffer*, where chunks are stored before playback, 2) the *knowledge record*, where information is kept about remote peers' presence, data content, past relationships, and current local node associations, and 3) the *trading logic*, whose role is to request chunks from neighbors and to choose and schedule the ones that are to be sent.

1) Data Buffer: Each node has a buffer where it collects and stores the data chunks prior to playback (Figure 1).

Definitions: The buffer uses a sliding window to regulate the stream reception. The sliding window is W chunks wide. Its goal is to output a stream of chunks with a desired maximum loss ratio. We call *buffer edge* the leftmost end of the sliding window. A second window, called *zone of interest*, lies on the left of the buffer edge and covers the chunks that will soon be needed as the sliding window moves. We refer to the sequence of chunks covered by these two windows as the peer's *trading window*, since it contains all chunks the peer is currently trying to obtain through exchanges with neighbors.

We define by *instantaneous position* of a node in the system, referred to as $T_{B_{inst}}$, the lag of its buffer edge from the source. This value can fluctuate quickly, so nodes keep a running average of their instantaneous position, previously referred to as $T_{B_{avg}}$, to filter the short-term position variability due to the unpredictable delays of the data exchange process. As above, T_D is the fixed lag after which a chunk can be discarded.

Moreover, we define T_Q as the interval of chunks ranging from the end of the sliding window to the chunk being currently played. The play-out delay T_V is initially set as $T_V = T_{B_{inst}} + W + T_{Q_{init}}$ after enough data chunks (to fill at least the configurable $T_{Q_{init}}$ interval) have been gathered. As $T_{B_{inst}}$ is free to change and since T_V remains constant (until the peer either disconnects or suffers from buffer under-run), T_Q is then equal to $T_V - (T_{B_{inst}} + W)$. T_Q's function is twofold: it grants an initial safety margin against variations of $T_{B_{inst}}$over time, and the changes in its size can be used by peers to evaluate their current data-reception stability.

Operation: A *sliding tolerance* parameter S defines the minimum amount of chunks that have to be present inside the sliding window before it can move forward. The maximum chunk loss rate tolerated during normal peer operation is thus bound by $LR = 1 - \frac{S}{W}$. The system-wide parameter LR_{max} is equal to the amount of redundant coding performed by the source. The value of S at any peer must be set so that $LR \leq LR_{max}$ to ensure the complete recovery of the original stream, but peers with enough bandwidth resources can obviously decrease their LR tolerance at will.

If less than S chunks are available, the sliding window cannot move. The lag of all the chunks it contains increases as time passes and as new chunks are generated. In this situation, the window keeps drifting on the lag axis (to the right of Fig. 1) and $T_{B_{inst}}$ grows at constant speed. Only when at least S chunks have been collected, the window is allowed to slide and to reduce its $T_{b_{inst}}$ (to the left of Fig. 1). The window will then keep sliding as long as it contains at least S chunks.

Over time, $T_{B_{avg}}$ will be either decreasing, if the window is sliding at a higher average speed than the source generates chunks, or growing, if the window is stuck waiting to fill a gap or sliding with a lower speed.

2) Knowledge Record: The strict timing constraints on the data retrieval process emphasize the central importance of the concept of node position in the system.

In PULSE, the position of a node carries two pieces of information: an explicit one, that is the range of chunks a node is able to serve, and an implicit one, that is an estimate of the peer's trading capabilities related to the incentive mechanisms. Intuitively, cooperating peers will be able to receive new chunks faster than selfish ones, and thus will find themselves nearer to the source.

Node decisions are based on the current locally available knowledge, which includes:

- information gathered through direct measurements of network parameters (RTT, data throughput)
- advertisements about the address and buffer delay range $[T_{B_{avg}}, T_D]$ of other peers. These low-priority messages can either be exchanged among peers using gossip/epidemic protocols such as SCAM [12], or distributed

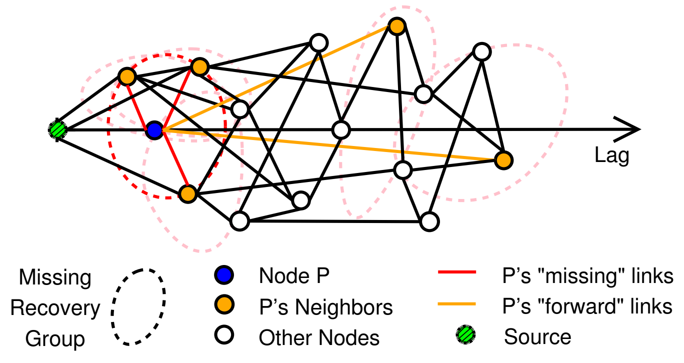


Fig. 2. A PULSE peer and its exchange sets (MISSING and FORWARD)

upon request by a centralized 'stream tracker' (similar to a BitTorrent tracker). Nodes known with this level of detail can be selected as targets for data exchanges.

- detailed accounts of the exact content of remote peers' buffers. These high-priority messages contain the instantaneous node position $T_{B_{inst}}$, T_D, a bitmap summarizing the chunks present in the trading window, and (optionally) explicit request/denial bitmaps for chunks in that range. Usually, nodes known with this level of detail are currently engaged in data exchange with the local peer.
- local records of previous trading interactions, in the form of a cumulative history score H.

The peers we are trading data with fall into two groups, MISSING and FORWARD (Figure 2). Peers in the MISSING exchange set are chosen among the "neighbors", whose trading window is overlapping to ours. Data exchanges with them can thus be mutual, if both sides have chunks the other one needs. These partners are likely to provide most of the chunks the local peer needs, as long as the local peer reciprocates. FORWARD exchange partners, on the other hand, may have a non-overlapping trading window. This means that, while the local peer can provide them with data, they cannot give back because all the data they can offer is already present in the local buffer. The FORWARD peer set helps introduce a component of altruism in the system and, at the same time, allows resourceful nodes to contribute more of their serving capacity to the system.

3) Trading Logic: The trading logic controls all the aspects of chunk request, selection, and scheduling. It processes the information coming from both the local buffer and the knowledge logic to decide which chunk will be sent to which neighbor.

III. Algorithms

The algorithms presented in this section make up the core of the PULSE system. They determine how each node chooses its partners for data exchange, how chunks to be sent are chosen and scheduled, and which chunks are to be requested from each neighbor. The following algorithms are all based on the assumption that peers *a)* have some knowledge of the other peers in the system, acquired through the normal mechanics of information exchange, *b)* can determine their position in the stream with respect to the media clock, and *c)* know or can estimate the maximum outbound bandwidth they will be able to provide.

A. Peer Selection

The main algorithms used for peer selection are an altruistic tit-for-tat algorithm similar to the one used in BitTorrent, and a simple cumulative trust metric. They are both executed at the start of each EPOCH, and give as result two lists that contain peers to which data will be provided.

1) History Score: Every node maintains a record of the previous interactions with every other peer as a numeric value, which we refer to as the *history score*. This mechanism enables a peer to build a knowledge base about its fellow peers: its goal is to gather data on past behavior that will allow nodes to make *informed choices* when selecting future candidates for FORWARD exchanges.

The algorithm is the following: each time a previously unknown peer is encountered, it is given an initial positive score. The node's score is incremented by some fixed value whenever a useful chunk is received from the node while it is not present in any of the local exchange lists. The node is subsequently added to a temporary exchange list which is cleared at each EPOCH. The node's score is decreased by some fixed quantity whenever it is chosen as FORWARD partner and receives one or more chunks from our peer.

At the moment, the history mechanism is quite simplistic, and we are aware of this. We believe that a lightweight trust metric similar to the one used in GnuNet [14] could be successfully applied to our system, improving the quality and strength of durable relationships, especially among the more resourceful nodes. We leave this aspect for future work.

2) Bandwidth Allocation: At any given moment, each peer must maintain several connections for sending and receiving data. Peers can limit the number of connections they establish, but network parameters such as link latency and data throughput depend on external factors that peers cannot control.

To simplify the problem of bandwidth allocation, PULSE peers reserve a fixed number of *connection slots* for data exchange. Since today node bandwidths are often asymmetric (with the outgoing bandwidth being much smaller than the incoming) it is mainly important to regulate the number of outbound connections.

A small number of connection slots (e.g. four) should be reserved for MISSING exchanges: this number should be chosen so that a peer with barely sufficient bandwidth resources can attain a reasonable theoretical throughput on each connection (e.g. SBR/4). Increasing the number of MISSING slots will increase the network and computational overhead, while improving the odds of quickly finding a useful chunk.

A variable number of slots can then be assigned, depending on the available outgoing bandwidth, to FORWARD exchanges. These slots will allow resourceful peers to contribute their excess bandwidth to the system by providing other peers with chunks without expecting an immediate return.

3) MISSING *List:* All the peers that sent us data during the last epoch are ordered by the number of non-duplicate chunks we received from them. A configurable quota of MISSING exchange slots is then filled with the highest-ranked nodes. When one or more MISSING slots remain available, they are allocated *a)* to known nodes with a trading window overlapping to ours, sorted by decreasing overlap size (the network latency bias may also be taken into account), and *b)* to randomly selected known nodes. Random selection is mainly used during the bootstrap phase.

4) FORWARD *List:* Peers are ordered by decreasing history score, and selected only if their trading window is not overlapping with the local trading window (i.e., the remote node is currently farther from the source than the local peer). Nodes already belonging to the MISSING list are discarded.

5) Source: The source differs from the other nodes since it doesn't need to engage in exchanges to get data chunks. It always has a complete sliding window, and its lag value is zero by definition. As a consequence, the peer selection algorithm at the source also needs to be different.

Moreover the source, lacking the data exchange feedback mechanism, could be exploited by malicious nodes that try to retrieve all chunks directly. The attackers could thus avoid contributing to the system and may also put in danger the entire distribution process if the source's bandwidth is small. To mitigate this danger, the source must change the subset of nodes it serves at each EPOCH and must avoid sending groups of contiguous chunks to the same peer.

The peer selection algorithm we employ is similar to the one used by the seeds in the latest BitTorrent software versions [15]. At the beginning of each EPOCH, the source prepares a list of known peers that have a T_B value (instantaneous or average) smaller than a fixed threshold. It then chooses randomly a small subset to which it will send chunks during this EPOCH. The source treats this list as its MISSING list.

B. Chunk Selection: Sending

A good chunk selection strategy is one that distributes the chunks in an uniform way across the nodes. It should also ensure that the buffer content of nearby nodes is different enough that they can engage in mutual transactions and concurrently exploit their multiple connections. Finally, it should also take into account remote peer's requests, which prevent duplicate chunks from being transmitted by several neighbors.

For all kinds of data exchanges, the chunks to be sent are selected comparing the requests received from each peer (or its whole buffer bitmaps, if a request was not specified) to the chunks currently held in the whole local buffer. The selected chunks are then sorted using some ordering criteria, and the first one is chosen for sending. It is indeed possible to queue several chunk uploads toward the same peer to benefit from the effects of transfer pipelining.

The criterion we are currently using for ordering chunks is a "Least Sent First, Random" strategy. Each peer keeps a counter of how many times it has sent each chunk. The one that has been sent the least number of times is chosen. In case of a tie, one of the chunks is selected randomly.

This scheduling strategy showed encouraging results in preliminary simulations, since the newest/rarest chunks to be received are among the first that are sent, getting a higher priority than chunks that have already been replicated. Breaking ties with a random choice (instead of, for example, by selecting the chunk whose lag is lowest) aims to avoid the preferential replication of the same chunks that may happen in situations where several peers have their buffer windows synchronized.

C. Chunk Selection: Requesting

The algorithm for chunk requests is similar to the heuristic used in DONet/CoolStreaming. Its purpose is to request the rarest chunks among those that are locally available, and to distribute the requests across different possible providers.

Using the local knowledge gathered from the MISSING neighbor set, chunks that are rarest across the neighborhood are requested with higher priority than more common ones. Chunks with the same number of providers are randomly requested from one of the neighbors that can provide them. To limit the load on any single peer, a maximum limit of requests per node is set.

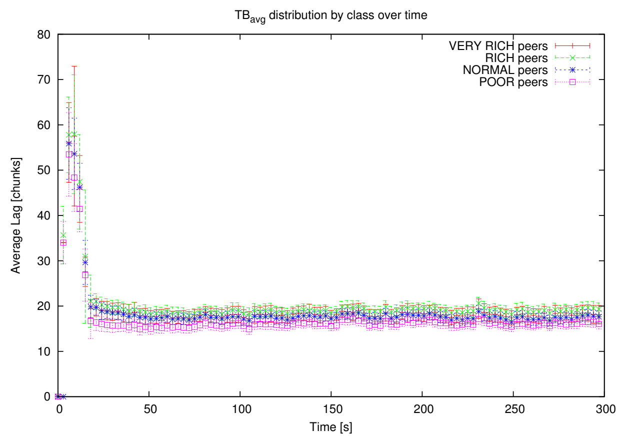
The combined effect of the sending and requesting policies appears to be satisfactory in our preliminary simulations. The information of the local rarity of a chunk, as seen by the requester, is implicitly conveyed to the sender through the request itself. Receiving a previously requested chunk, on the other hand, enables a peer to propagate with high priority a chunk which is still supposed to be rare. The evaluation of alternative chunk selection schemes is left as a subject for future work.

D. Simulation Results

We performed simulations of the algorithms' behavior for different network sizes, bandwidth distributions, buffer parameters, and node arrival patterns. While we cannot include a full discussion about our results in this paper, we will briefly illustrate the most striking global properties.

Our simulations are performed using a simple round-based simulator. The simulated network has a single-stub topology. Nodes are connected to the stub through access links whose bandwidths are configurable. Bandwidth allocation is performed using a slot-based mechanism: each slot allows the transmission of one data chunk between two nodes during one round. Network latencies are not taken into account in the model: control information is propagated without delay and the only source of latency is due to data transfers. While the information about updated buffer delay ranges is available to everyone, a node's detailed buffer situation is only known to the nodes that have the node in their exchange lists or that received messages from the node during the current EPOCH.

Figure 3 illustrates two five-minute traces for a network of one thousand nodes. Four bandwidth classes with strongly heterogeneous outbound capacities (from $SBR/2$ to $10 \cdot SBR$) are present. The first trace is obtained with all nodes joining



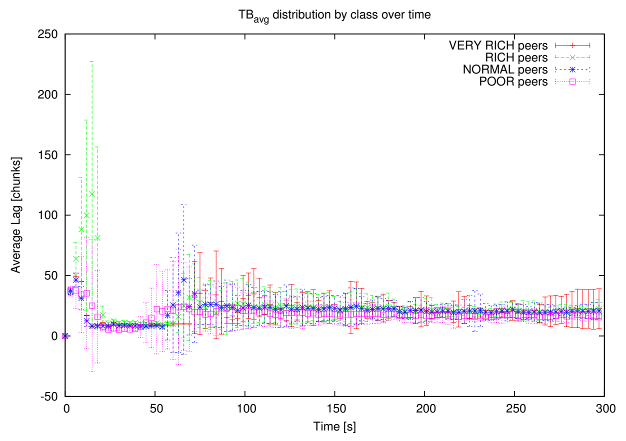


Fig. 3. Simulation traces for a PULSE network of 1000 peers (4% "very rich" peers with symm. bandwidth of $10 \cdot SBR$, 21% "rich" peers with symm. bandwidth of $3 \cdot SBR$, 20% "normal" peers with asymm. bandwidth of SBR UL and $2 \cdot SBR$ DL, and 55% "poor" peers with asymm. bandwidth of $SBR/2$ UL and $2 \cdot SBR$ DL). $SBR = 16$ chunks/s. Top: all nodes join at $t = 0s$ - Bottom: flashcrowd arrivals around $t = 120s$.

simultaneously at $t = 0s$, while the second shows the effect of a flashcrowd-like arrival pattern, with peak arrival rate around $t = 120s$. In both cases, after a short initial instability, the system converges to a small average delay value, which is approximatively 20 chunks. The trading window size for these runs was set to 64 chunks, i.e. 4 seconds worth of stream data. Instability is mainly due to the randomness of the arrival process: the nodes that didn't manage to connect at the first attempt have to disconnect before retrying.

In the second trace we can observe the response of a running system to the addition of a substantial number of nodes in a short time (more than 500 nodes join between $t = 60s$ and $t = 180s$): while the class delay averages slightly increase, the delay variance of the poorest classes is especially impacted. Intuitively, we can see that the role of tit-for-tat incentives is mostly important when resources become locally scarce, as happens when many new nodes join or when some rich nodes quit the system. In these cases, more resourceful nodes will manage to obtain a better position (or to avoid performance degeneration) with respect to the others. Otherwise, the altruistic allocation of excess resources tends to prevail, and produces a system with homogeneous performances, regardless to individual bandwidth contribution.

IV. Conclusions

In this paper we presented PULSE, a P2P system for live media streaming. Our system's innovative features are the presence of strong incentives to share combined with the absence of a fixed overlay structure. Nodes try to reduce their lag from the source by trading chunks and selecting cooperative neighbors for lasting relationships.

The resulting PULSE topologies dynamically adapt to the available bandwidth resources. The system can work in a wide range of conditions since it does not put a lower bound on the contribution required from each node. Moreover, especially when overall outgoing bandwidth is scarce, the more resourceful nodes tend to stabilize on lower lag values and to obtain chunks from the source earlier. This adaptive node placement scheme confers a good scalability to the system.

We currently perform extensive simulations of PULSE for various scenarios to gain a better understanding of the impact of local settings on global dynamics. As a next step, we will implement a full-fledged prototype of PULSE. The long-term focus of our work is the large-scale deployment of our system.

References

[1] K. Sripanidkulchai, A. Ganjam, B. Maggs, and H. Zhang, "The Feasibility of Supporting Large-Scale Live Streaming Applications with Dynamic Application End-Points", in *Proc. of ACM SIGCOMM '04*, Portland, Oregon, OR, USA, August 2004

[2] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "SplitStream: High-Bandwidth Multicast in Cooperative Environments", in *ACM SOSP*, Bolton Landing, NY, USA, October 2003

[3] H. Deshpande, M. Bawa, and H. Garcia-Molina, "Streaming live media over a peer-to-peer network", in *Work at CS-Stanford*. Submitted for publication, 2002

[4] H. Deshpande, M. Bawa, and H. Garcia-Molina, "Streaming live media over peers", Tech. Rep. 2001-31, CS Dept., Stanford University, 2001

[5] Y. Chu, A Ganjam, T. S. E. Ng, S. G. Rao, K. Sripanidkulchai, J. Zhan, and H. Zhang, "Early Experience with an Internet Broadcast System Based on Overlay Multicast", Technical Report CMU-CS-03-214, Carnegie Mellon University, December 2003

[6] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast" in *ACM SIGCOMM*, Pittsburgh, PA, USA, 2002

[7] D. A. Tran, K. A. Hua, and T. T. Do, "A Peer-to-Peer Architecture for Media Streaming", in *IEEE JSAC*, vol. 22, no. 1, January 2004

[8] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum, "CoolStreaming/DONet: A Data-driven Overlay Network for Live Media Streaming", in *IEEE INFOCOM'05*, Miami, FL, USA, March 2005

[9] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh", in *ACM SOSP*, October 2003

[10] T. Nguyen, and A. Zakhor, "Distributed Video Streaming with Forward Error Correction", in *Proc. of Packet Video Workshop*, Pittsburgh, USA, April 2002

[11] V. K. Goyal, "Multiple description coding: Compression meets the network", in *IEEE Signal Processing Magazine*, September 2001

[12] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié, "Peer-to-peer membership management for gossip-based protocols", in *IEEE Transactions on Computers*, Vol. 52, No. 2, February 2003

[13] B. Cohen, "Incentives Build Robustness in BitTorrent", in *Proc. of Workshop on the Economics of P2P Systems*, 2003

[14] C. Grothoff, "An Excess-Based Economic Model for Resource Allocation in Peer-to-Peer Networks", in *Wirtschaftsinformatik* 3-2003, June 2003

[15] A. Legout, G. Urvoy-Keller, and P. Michiardi, "Understanding BitTorrent: An Experimental Perspective", *Tech. Report* inria-00000156, INRIA, Sophia Antipolis, November 2005.