

On The Feasibility Of Centrally-Coordinated Peer-To-Peer Live Streaming

Roberto Roverso^{1,2} Amgad Naiem^{1,3} Mohammed Reda^{1,3} Mohammed El-Beltagy^{1,3} Sameh El-Ansary^{1,4}
Nils Franzen¹ Seif Haridi²

¹ Peerialism Inc., Sweden, ² KTH-Royal Institute of Technology, Sweden,

³ Cairo University, Egypt, ⁴ Nile University, Egypt

{roberto, amgad, sameh, mohammed}@peerialism.com

Abstract—In this paper we present an exploration of central coordination as a way of managing P2P live streaming overlays. The main point is to show the elements needed to construct a system with that approach. A key element in the feasibility of this approach is a near real-time optimization engine for peer selection. Peer organization in a way that enables high bandwidth utilization plus optimized peer selection based on multiple utility factors make it possible to achieve large source bandwidth savings and provide high quality of user experience. The benefits of our approach are also seen most when NAT constraints come into play.

I. INTRODUCTION

Peer-To-Peer live streaming (P2P IPTV) is a challenging problem that has attracted the focus of many academic and industrial research communities. Debates have been going on about two rival approaches: mesh-pull and tree-push [1]. Hybrid [2] and best-of-both-worlds mesh-push [3] have also been investigated. That said, some still see that despite current deployments and reported successes, P2P IPTV is still at its early stages [4]. In this paper, we report our experience with a third approach to P2P IPTV where a central entity plays a bigger role in overlay structuring. Central entities in P2P systems are a taboo from a P2P purist’s perspective, but are major components of prominent P2P systems such as the tracker in Bittorrent. Previous works [5] aimed to make Bittorrent’s tracker or the seed more intelligent [6]. Central coordination in our case could be perceived as an attempt to take tracker’s role to the limit. The main challenge we faced was to design an efficient optimization engine which can provide decisions in a very short period of time, or else the outcome might be of no value. If the overlay network changes considerably due to peer dynamics while the optimization engine is running, then connectivity recommendation of the engine might even be detrimental to the system’s performance. Our main contribution is that we show that a system employing such an approach is feasible and that the required central computing resources are not by any means prohibitive. Subparts of this system have been

published before as generic components such as the idea of our optimization engine [7] and its parallelization on GPUs [8] as well as our NAT connectivity [9], but this is the first time we describe how these subparts work together.

II. SYSTEM ARCHITECTURE

The main entities in the system are: *i*) Clients (peers) who want to watch the live stream and are normally behind home or corporate NAT gateways, *ii*) Streaming source connected to the streaming server but otherwise exactly like any normal peer, *iii*) The tracker which centrally coordinates the system, *iv*) The optimization engine which has a snapshot of the overlay and handles joins, failures and restructuring of the overlay, *v*) Connectivity server, to facilitate connection establishment between peers behind NAT, *vi*) Bandwidth measurement server that peers use to get an approximate guess of their upload capacity.

A typical scenario for a client is as follows: The client contacts the bandwidth measurement server to estimate its upload capacity and then requests a video stream from the tracker providing information about its upload bandwidth. The tracker forwards the request to the optimization engine which selects providing peers for the requesting client. The tracker notifies both the requesting and the providing peers involved in the operation. The providing peers will then start to push the stream to the requesting peer after using the connectivity server to traverse NAT gateways if needed. Periodically, the optimization engine restructures the overlay and reconfiguration orders are sent to the clients.

III. TERMINOLOGY: SEATS & PERSONS

We assume that a stream is divided into a number of stripes. For instance, if the stream rate is 1 Mbps, and we used 4 stripes, each stripe would be a sub-stream of 256 Kbps. Given a peer with an upload capacity of 1.5 Mbps, we say that this peer has 6 “seats” because it can upload to other peers 6 stripes simultaneously. Each client will have to find a seat for the 4 stripes, so we say that he has to feed 4 “persons”. That is, we discretize the

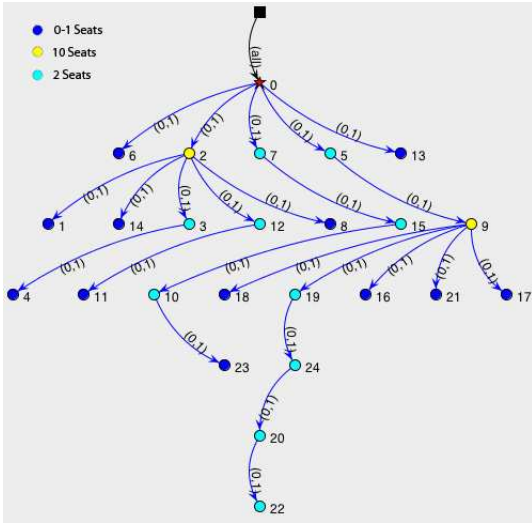


Fig. 1. Overlay before reconfiguration

upload capacity to units called seats and the download capacity to units called persons. It is the task of each of the client to request seats for each of his persons. Such division of bandwidth into persons and seats is made so that the optimization engine can have a simple model of the bandwidth/download capacities of the peers.

IV. OVERLAY STRUCTURE

Basic peer Joins. Peers with different number of seats join the network in an arbitrary order. The most basic join mechanism is to start by seating the peers on the source seats until, eventually, all the source seats are occupied. At this stage, we say that the first row is full and the seats of the joined peers form a second row. New joiners are seated at the second row until it is full and in their turn they form a third row and so forth. Naturally, the order of joins matters. The extreme worst case would be that a number of peers, all with zero seats, come and occupy the first row with no second row created. The extreme best case would be that peers with large number of seats join before the peers with less seats. In reality, we get some order that is somewhere between the two extremes. The tracker periodically runs a reconfiguration process and tries to bring the overlay into the best possible state.

Reconfiguration phase 1: row construction. The process starts by sorting all the peers according to their seat count in descending order. The peers with high seat count are seated at the first row and the process continues as above. This leads to maximum upload bandwidth utilization and minimum number of rows, which directly translates into smaller playback delays. At this stage, we are sure that every row can provide enough seats for the rows below, and the problem now reduces to the assignment of persons of every row to seats of the row above. At this point, even a random assignment

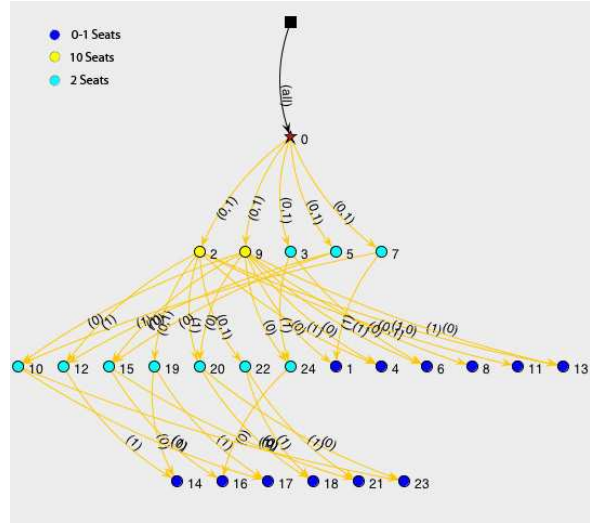


Fig. 2. Overlay after reconfiguration

between peers can be carried out because an important part of the decision process has been done already by sorting the peers and compacting the rows. We explain next, how a more optimized assignment is achieved. To avoid confusion, we stress that this reorganization happens first at the tracker internal snapshot of the overlay, then a batch of messages is sent out to rewired the peer connections. In Figure 1, Figure 2 we show an example of the overlay before and after reconfiguration respectively.

Reconf. phase 2: happiness matrix

The different assignment combinations between persons and seats form columns and rows of a matrix respectively. This matrix we call the ‘‘Happiness Matrix’’ $A(i, j)$, which represents all possible interconnections between peers in two consecutive rows and whose values express the worthiness of such connections. A ‘‘Happiness Value’’ a_{ij} in the matrix is a weighted sum of all characteristics observed and/or expected from a certain connection between person i and seat j . The weighted characteristics for this happiness value are: a) *Inter-peer delay*, peers with lower delay are favored, b) *Stickiness*, connections that already exist are favored to minimize interruptions, c) *Buffers matching*, seats with buffers who have data that is useful to the persons are favored, where usefulness is proportional to how much the seat of the uploading peer is more advanced in the sub-stream compared to the person of the downloading peer, d) *NAT Compatibility*, defines the probability of a connection to succeed between two peers. A value is given accordingly such that higher value means a higher probability of connection. We elaborate on NAT issues hereunder shortly, e) *ISP Friendliness*, peers in the same autonomous system (AS) are favored. In fact, we have an engine which computes the hop count between ASs

and the smaller distance is favored.

The resulting value can be considered as a grade for a certain combination. If many uploaders are available for a certain peer to download from, the one with highest grade of all the uploading peers will be chosen. Which means that, for a certain person A , a seat B will be selected which is expected to provide the best performance in the future transfer between A and B . Such mechanism of assigning persons to seats has been modelled as a Linear Sum Assignment Problem. The task of solving the optimization problem is carried out by the Optimization Engine. Once a result is produced, the tracker notifies the peers so that the new transfers can be established.

There are two sensible steps in this process: the calculation of the ‘‘Happiness’’ values and the actual solving of the Optimization Problem. In the first case, it’s trivial to understand that the choice of the happiness values will directly impact the performance of the system. For instance, one of the parameters in the calculation of the Happiness values is the Stickiness and the value chosen might result in more stable system but the overall bandwidth utilization might be affected, since the system will give higher priority to the preservation of successful connections than to load balancing. Thus a careful calculation of the values of the $A(i,j)$ matrix is developed as the choice of weights of different characteristics affecting the happiness value.

Reconf. Phase 3: Solving the assignment problem.

The last step in the Optimization process is the actual solving of the linear optimization problem between pairs of rows to assign seats to persons. In fact, the computation associated with it might take a long time to execute, since the number of potential peer combinations is typically quite large. In the presence of high churn, disruptions in the network which happen as the optimization is taking place might totally change the validity of the initial information which the ongoing computation is based upon. This will cause the results of the calculation to be of limited or no value. It’s therefore vital for the calculation to happen as fast as possible to avoid such situations. For this purpose, initially we used the Auction algorithm [10] to solve the optimization problem, which is known to be one of the fastest algorithms for solving complex Linear Sum Assignment Problem (LSAP). However its performance fell very short of our needs given the size of the problems to be solved. Consequently, we have developed a new heuristic solver based on a local-search approach called *Deep Greedy Switching (DGS)* which has been published in [7]. It sacrifices very little in terms of optimality, for a huge gain in the running time of the algorithm over other

methods. The DGS algorithm provides no guarantees for attaining an optimal solution, but in practice we have seen it deviate by less than 0.6% from the solutions reported by the auction algorithm. Such a minor sacrifice in optimality is acceptable in our system where speed is the most important factor as an optimal solution that is delivered too late is practically useless. Compared with the auction algorithm, DGS has the added advantage that it starts out with an initial assignment and keeps improving that assignment during the course of its execution. The auction algorithm, however, attains full assignment only at termination. Hence, if a deadline has been reached where an assignment must be produced, DGS can be interrupted to get the best assignment solution it has attained so far. We were also able to parallelize the DGS heuristic on commodity GPUs [8] and solve instances of 10,000 peers overlays in less than 3 seconds.

Churn. Beyond the basic join described above we use some other techniques. For instance, in each row we reserve some slack seats for future use. This helps us, during the peer joins and failures. During the join process we try to put a peer not at the last available row as described in the basic join above, instead we try to predict which row the peer will be at should a reconfiguration take place. This results in peers with high seat count ending up closer to the root of the tree even if they came late. Moreover, it reduces the possibility of disruption caused by reconfiguration as we place a new peer in its deserved row directly. If for any reason the reserved seats were all occupied we revert to basic join. If that also did not help, due to lack of seats in the last rows, we reserve what we call fallback source seats, and if these were also totally occupied, we have one final strategy which is a waiting list where the peer has to wait until the next reconfiguration to be admitted to the network when some seats later become available. For failures, the person facing the failure of his providing seat will try to fallback to the slack seats of the row above. If there is not enough slack, the slack of a higher row are attempted. In a sense, handling a failure is like a partial join process but not necessarily for all persons of a peer. Churn usually reduces the optimality of the overlay and reconfigurations take care of bringing it back in shape.

NAT Heuristic. With NAT in the picture, the *effective* upload capacity of a row, is determined by the compatibility of the NAT types of the peers in it with the ones in the row below. For instance, a client behind a very restrictive NAT can have a huge upload capacity but effectively it is much less because it can only be used to upload to a limited subset of peers. In our previous

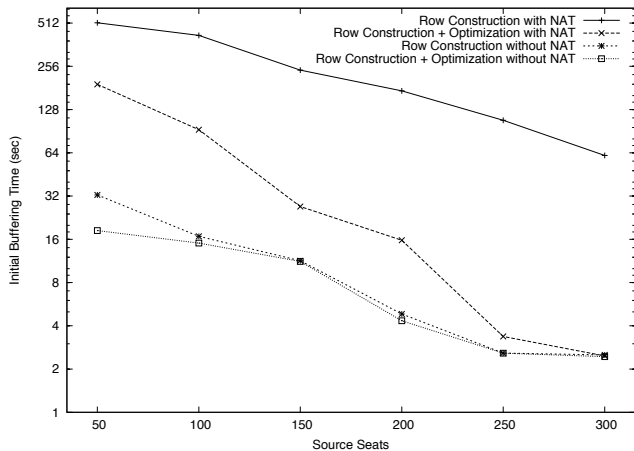


Fig. 3. Initial buffering time

work entitled “NATCracker”[9], we described a rigorous classification of the various behaviors of NAT gateways beyond the traditional classification of only four types. We have also provided an analysis of which traversal strategy should be used to connect peers to each other according to their types. In this context, we make use of this model to find an optimal placement of peers in rows in order to satisfy the download demands of as many peers as possible while maximizing pairwise connectivity between them. In order to achieve such trade-off, we formulated a heuristic based on the max-flow approach[11], where s is the source of the flow (our streaming server) and t is a virtual sink node collecting all spare capacity in the network. The heuristic works as follows: first, we carry out row construction as previously described. For each row r , we aggregate peers in sets N_t^r according to their NAT type t . Each set N_t^r is considered as a virtual node in the max-flow network. Each virtual node has a cumulative capacity of $u_{N_t^r}$ number of seats, i.e. the sum of all seats for the peers in the set. Then, we establish edges between a set $N_{t_i}^r$ in the current row and all sets $N_{t_j}^{r+1}$ in the row below it, such that t_i is compatible with t_j . The aforementioned process is carried out for all the rows in the tree. When this placement process is completed, we execute a standard max-flow algorithm to push as much flow as possible from the source s to the sink t through the virtual nodes. After that, we proceed to switch peers between rows according to their outbound and inbound flow. For instance, we choose the set $N_{t_i}^r$ with the least flow in a row and we identify which has the weakest upload bandwidth in it. We then swap this peer with the one in the row below which has the biggest upload capacity but whose NAT type t_c is different from t_l . We then run the max-flow algorithm again. The process is repeated multiple times until each row has enough bandwidth capacity to provide for the row below it given

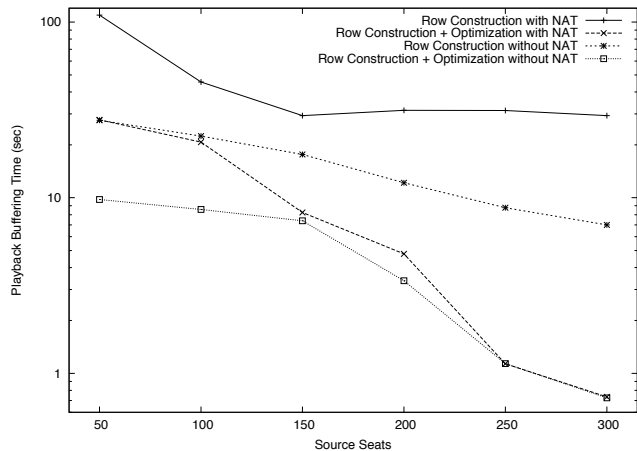


Fig. 4. Playback Delays

the connectivity constraints in place between provider and receiver peers.

V. SIMULATION RESULTS

We present a simulation of our system over a discrete-event simulator. It is worth mentioning here that we tried to make sure we model bandwidth accurately[12]. In addition NAT semantics with real-world probability of connection establishment and type encountering probability are also simulated based on the NATcracker work [9]. We simulate 1000 peers watching a 30 minutes stream. Half of the network is there when the stream starts and the rest join with average rate of 2 peers per second. One tenth of the network fails during the stream playback. The streaming rate in 600 Kbps divided into two stripes. The seat distribution is 10% : 10 seats, 40% : 2 seats, 40% : 1 seat and 10% : 0 seats. We also have to stress that the scalability of our simulation is not limited by the optimization engine but rather by the simulator’s modelling of peer bandwidth allocation and deallocation. The simulation is done as follows, we measure the performance of our approach with and without optimization after row construction. However, we consider the NAT heuristic as part of the optimization process. Then we repeat the experiment by solving an easier problem where peers are not behind NAT. The idea is to show how much the optimization contributes to user experience and show that when one solves the problem in a network where peers are behind NAT in place, the problem is much harder. Additionally, we want to see how much of the problem is solved by row construction alone and how much depends on the optimized assignment.

Initial buffering time. This is the time before stream playback starts. As shown in Figure 3, in the absence of NAT constraints, and as long as a peer can find available seats, playback can start immediately. Therefore optimization is not the key element. However, with NAT

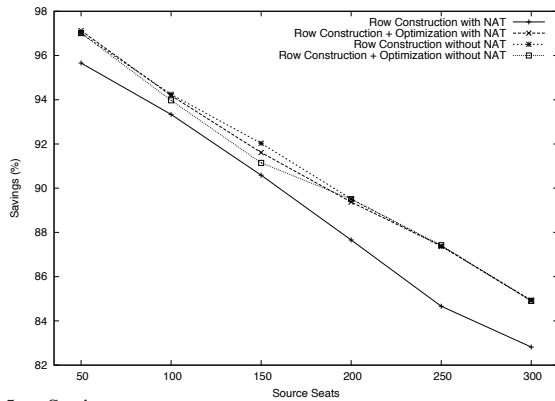


Fig. 5. Savings

constraints, a peer might find available seats but can not use them because the seat and the person NAT types might not be compatible. Thus, persons will be randomly assigned to NAT-incompatible seats, the connection will fail, and the person will try rejoining until, eventually, getting admitted. That's why maximizing the probability of successful connection plays a very significant role and that is something that optimized assignment provides and in its absence, the initial buffering time is ridiculously higher by orders of magnitude. The number of source seats can drastically affect the results as well, because in general it is a last resort for peers with severely constrained NATs. When enough source seats are provided, the optimization process can bring the initial buffering time of a network with NAT constraints to one without NAT constraints. However, in all cases, we varied the source bandwidth between 10 and 100 Mbps.

Playback delay. This is the sum of the time periods where playback is paused due to the buffers running out of content. As shown in Figure 4, with or without NAT constraints, in the lack of optimization, the performance lags by at least one order of magnitude. We also see that even in the presence of NAT constraints and with some extra help of source seats, the optimization can bring the user experience to one that is close the experience of a network without NAT constraints.

Source bandwidth savings. In Figure 5 we can clearly see a trade-off between savings and quality of experience. However, with a user experience where we have around 2.5 seconds of initial playback delay and 2 seconds of playback delay, the savings are at around 85%. More importantly one can see that row construction is the primary driver of savings not optimized assignment.

VI. SCALABILITY DISCUSSION & CONCLUSION

We have reported in this paper our experience with tackling the problem of P2P live streaming using central coordination. We were able to provide a feasible solution by using a heuristic linear sum assignment problem

solver capable after parallelization on commodity GPUs to handle 10,000 peers overlay in less than 3 seconds. The merits of the solution is high bandwidth source savings due to high bandwidth utilization of the peers, low initial buffering time and playback delays. The main point of our approach compared to other decentralized approaches is that we try to let a central entity help the peers in the peer selection process, avoiding with that, the trial and error process for discovering the best overlay configuration. Beyond the scale of 10,000, one can partition the network into multiple trackers. Another approach is to manage a backbone of nodes using central coordination and let a group of swarming peers stream from each backbone node. The approach is generic enough to be modified for other problems. For future work, we are considering replica placement in distributed storage as well as a NAT traversal facilitator for generic overlays.

REFERENCES

- [1] F. Picconi and L. Massoulie, "Is there a future for mesh-based live video streaming?" in *Peer-to-Peer Computing*, 2008. *P2P '08. Eighth International Conference on*, 2008.
- [2] F. Wang, Y. Xiong, and J. Liu, "mtreebone: A hybrid tree/mesh overlay for application-layer live video multicast," in *ICDCS '07*, Washington, DC, USA, 2007.
- [3] R. J. Lobb, A. P. Couto da Silva, E. Leonardi, M. Mellia, and M. Meo, "Adaptive overlay topology for mesh-based p2p-tv systems," in *NOSSDAV '09*, 2009.
- [4] X. Hei, Y. Liu, and K. W. Ross, "Iptv over p2p streaming networks: the mesh-pull approach," in *Communications Magazine, IEEE*, March 2009.
- [5] A. R. Bharambe, C. Herley, and V. N. Padmanabhan, "Analyzing and improving a bittorrent networks performance mechanisms," in *IEEE INFOCOM 2006*.
- [6] F. Esposito, I. Matta, P. Michiardi, N. Mitsutake, and D. Carra, "Seed scheduling for peer-to-peer networks," in *8th IEEE International Symposium on Network Computing and Applications*.
- [7] A. Naiem and M. El-Beltagy, "Deep greedy switching: A fast and simple approach for linear assignment problems," in *7th International Conference of Numerical Analysis and Applied Mathematics*, 2009.
- [8] R. Roverso, A. Naiem, M. El-Beltagy, S. El-Ansary, and S. Haridi, "A gpu-enabled solver for time-constrained linear sum assignment problems," in *Informatics and Systems (INFOS), 2010 The 7th International Conference on*, 2010.
- [9] R. Roverso, S. El-Ansary, and S. Haridi, "Natcracker: Nat combinations matter," *Computer Communications and Networks, International Conference on*, vol. 0, pp. 1–7, 2009.
- [10] D. Bertsekas, "The auction algorithm: A distributed relaxation method for the assignment problem," *Annals of Operations Research*, vol. 14, no. 1, pp. 105–123, 1988.
- [11] A. V. Goldberg and R. E. Tarjan, "A new approach to the maximum flow problem," in *STOC '86: Proceedings of the eighteenth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 1986, pp. 136–146.
- [12] R. Roverso, M. Al-Agga, A. Naiem, A. Dahlstrom, S. El-Ansary, M. El-Beltagy, and S. Haridi, "Myp2pworld: Highly reproducible application-level emulation of p2p systems," in *Decentralized Self Management for Grid, P2P, User Communities workshop, SASO 2008*, 2008.