University of Zurich[UZH]

MASTER'S THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Incentives and Social Welfare in LiveShift : a Congestion Games - Based Approach

Simon Poltier
Zurich, Switzerland
Student ID: 04-809-273

Supervisor: Fabio Hecht
Date of Submission: September 23, 2011

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zurich, Switzerland

ifi

Master's Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zurich, Switzerland
URL: http://www.csg.uzh.ch/

Kurzfassung

In Peer-to-Peer (P2P) -Anwendungen wie das LiveShift Video-Streaming System sind alle Teile des Systems (Peers) voneinander unabhängig. Wenn das System Ressourcen an ihnen bietet, verhalten sich den Peers egoistisch; im Systementwurf sollte man das berücksichtigen.

Diese Master-Arbeit schlägt einen Spieltheorie-basierenden Ansatz zum Peer-Verhalten im LiveShift Peer-to-Peer Video-Streaming System vor. In diese Kurzfassung werden die Ziele, das entwickelte Systemmodell und die Resultate dessen beschrieben.

Ziele

Zuerst will man ein Peer-Verhaltensmodell entwerfen, der jedes Peer erlaubt, eine bestimmte Nutzenfunktion individuell zu maximieren; so ist ein eigennütziges Peer-Verhalten festgelegt. Dieses Modell soll auch das Verhalten beschreiben, der aus Sicht das ganze System zu einer optimalen Lage führt. Um diese optimale Lage zu definieren wird eine Leistungsmetrik gebraucht; die muss man auch bestimmen. Basierend auf diese Definitionen wird ein Anreizsystem vorgeschlagen, der unter eigennütziges Peer-Verhalten der System so nah wie möglich an den optimalen Zustand bringt. In eine zweite Phase muss die Fähigkeit des Ansatzes, die Leistungen des LiveShift-Systems zu verbessern, bewertet werden. Der Ansatz sollte unter optimales (oder ungefähr optimales – je nachdem, wie implementiert) und eigennütziges Peer-Verhalten bessere Leistungen als die bestehende LiveShift Verhaltensregeln (*policies*) bieten.

Modell

Das Peer-Verhalten Modell ist in zwei Hauptteile gegliedert, wo Download- bzw. Upload-Verhalten getrennt behandelt werden. In beide Teile werden Nutzenfunktionen festgelegt und begründet. Diesen Funktionen hängen an Netzwerküberbelastung zusammen. Eine Leistungsmetrik und eine Beschreibung des Optimalverhaltens sind auch präsentiert. Diese sind nur unter eingeschränkte Bedingungen gültig; insbesonders muss sich die Peer-Bevölkerung nicht ändern. Um in eine realistische Lage das Modell zu testen, werden auch Verhaltensregeln definiert, die in in einem dynamischen System anwendbar sind. Diese Regeln sind vom optimalen Verhalten abgeleitet, aber sind nicht selbst optimal;

sondern definieren sie ein Peer-Verhalten, der gut fürs gesamte System ist, im Gegenteil zum eigennützigen Verhalten.

Resultate

Um das Modell zu testen wird eine Simulation entwickelt. Die Experimente, die damit gelaufen werden, prüfen die Effekte von drei Variablen: Ansatz, Peer-Verhalten (gut oder eigennützig) und Änderungsgeschwindigkeit des Systems (d.h. (Nicht-) Anwesenheit von Flash Crowds). Die Resultate sind insgesamt positiv für den vorgeschlagene Ansatz, aber nur im Gutverhaltetes Fall ist die Leistungssteigerung genügend, um eine effektive Leistungssteigerung im LiveShift zu bestätigen, als die Simulation nur ungenau das tatsächliches Systembetrieb LiveShifts spiegelt.

Abstract

Peer-to-Peer (P2P) systems are characterized by the autonomy of their constituent parts, the peers. As such, when the system's goal is to provide a certain resource to peers, those will participate in the system so as to maximize the benefits they get from it. This is the case in the LiveShift video streaming system, and thus explicitly taking that type of behavior into account in the system's design is desired.

In this thesis, a game-theoretic model of peer behavior for LiveShift is developed, dividing peers' behavior between download and upload phases, modeled as separate games. The behavior of peers under static, deterministic conditions is discussed and its outcomes presented; payoff functions reflecting peers' preferences in both the download and upload games are proposed. From these elements, rules of behavior are developed to improve the system's performance, both in settings where peers are selfish and in those where they are well behaved. Then, a simulation is implemented to test the approach's effectiveness in various scenarios, varying the system's level of dynamicity as well as peer behavior. Results are positive overall, but only present strong evidence of improved performance in scenarios where peers are well-behaved. Because of the limitations of the simulation vs. the actual software, only in that well-behaved case can it be concluded that the approach provides a performance boost.

Acknowledgments

Optional

Contents

Chapter 1

Introduction

The object of this thesis is to study the Peer-to-Peer (P2P) video streaming system, LiveShift [9], using the notions of Game Theory (GT) to model this system's behavior. In this introduction, the need for such an analysis is first explained; an outline of the remaining chapters is then given.

1.1 Motivation

P2P systems are in general decentralized in their organization and their resource consumption [19]. This means that each participant in the system (peer) will make decisions with respect to consuming resources and allocating resources to other participants. These decisions will be geared towards the peer's individual goals. In LiveShift, peers want to acquire video streaming content in a timely manner; as an autonomous part of the system, an individual peer only tries to acquire content for itself, and has no particular interest in seeing other peers acquire content. However, actions of the individual will affect others' ability to reach their goals, e.g. by using certain resources in the system or by allocating upload bandwidth. This means that it is in a peers' interest to behave strategically, i.e. to account for other peers' reactions to their behavior when deciding on a course of action.

For the designer of a system such as LiveShift, the goal is to ensure that the system in its entirety performs as well as possible. This first requires a definition of what a well-performing system is, i.e. a *performance metric*; then, the designer must try to build or modify the system so that it encourages peers to behave in a way that maximizes that metric. To do so, the designer creates and implements an *incentive scheme*, i.e. a set of rules that, when applied by certain parts of the system, encourage peers to behave in a way that steers the system towards a selected global outcome.

There is extensive research on the topics of incentive schemes, cooperation and strategic behavior in decentralized P2P video streaming systems. The cases of Video-On-Demand (VoD) and live streaming are usually treated separately; LiveShift [9] is a hybrid system that allows both, using a mesh-pull P2P approach.

The incentive scheme [3] compatible with this setting extends direct reciprocity to allow a greater number of transactions to take place while discouraging free-riders, thereby making sure limiting resources are not withheld by peers. However, the allocation of these resources is not necessarily optimal from the point of view of the entire system's social welfare. Also importantly, it simply assumes that some peers are free-riders, while others are well-behaved; individual self-maximizing behavior is not assumed.

The present study will therefore analyze the strategic situation that self-maximizing peers face when allocating bandwidth, and the behavior that results from it. Based on that analysis, its main goal will be to further optimize bandwidth allocation by selfish peers through incentives, under the assumption that a mechanism to prevent free-riding (such as Tit-For-Tat [7] or Private Shared History [3] is in place.

1.2 Description of Work

The study includes a simple theoretical model of peer behavior in a static (without peers leaving or joining) P2P environment. In that model,
– an optimum in terms of system performance,
– the peer behavior leading to that optimum,
– a performance metric of the system, and
– the incentives that peers face in the system
are formally defined. From this static analysis, rules of behavior are derived, describing the upload and download behavior of peers in a dynamic system, in both the well-behaved and self-maximizing (selfish) cases. Based on these two cases, incentives are formulated in order to bring the typical case as close as possible to the well-behaved case (in terms of the metric). These incentives are modeled as rules of behavior for the *peercaster*, which is the peer that originally produces the stream in LiveShift. The peercaster is assumed to have a social welfare-maximization goal, and therefore will comply with these rules. Download and upload behavior are modeled separately, which allows a focus on the allocation of upload bandwidth, the limiting resource in most cases.

In order to measure expected metric values and other characteristics, a simulation of the model is implemented, providing a way to run experiments with different parameters. In each experiment, the overall system welfare is measured using the aforementioned metric. The expected results are that the behavior determined defined in the model should be an improvement over current LiveShift operation, and that selfish peer behavior with the countervailing incentives proposed in the model will present better performance than without those incentives. Since one of the goals is to have incentives that are well-adapted to highly dynamic systems, it is also expected that these advantages will be stronger in a less stable setting (e.g. in the presence of flash crowds).

1.3 Thesis Outline

The remainder of the thesis is organized as follows: first, background information on game theory and related work in the area of incentive schemes are presented; then, Chapter 3

deals with the theoretical model, its properties and interpretation into a dynamic set of rules; a description and discussion of a simulation of this model is then provided. Next, the implementation of the simulation is described. Finally, the results of the experiments are presented and evaluated.

Chapter 2

Related Work

This chapter provides an introduction to LiveShift, and then to game theory (GT). Finally, studies with goals similar to the present thesis' are discussed. These mainly comprise of the development and application of various incentive schemes for and to P2P streaming systems.

2.1 LiveShift

LiveShift [9] is a P2P live streaming application that allows time-shifting, e.g. for users who join a stream some time after it has started. It is based on the structured P2P library TomP2P[2]. The concept of channel is used to distinguish the different pieces of content being offered. Channels are then split into segments and blocks according to a certain time scale. Segments are the content unit whose availability is advertised by peers, whereas blocks are the smallest unit of content that can be validly transferred, and availability information for them is only sent when a peer requests a segment.

A mesh-pull approach is used: actual content has to be requested by peers who want to receive it (only metadata, i.e. possessed blocks, is pushed); and there is no formal hierarchy among peers, although peercasters (those who originally provide streams) may behave differently. Also of note is that there are two types of information stored in the same address space: channel IDs (for users to be able to choose a channel) and available segments.

The behavior of peers is determined by policies, i.e. choices that peers make with respect to the way they behave toward one another. Those policies include block selection (how far ahead should a peer download), candidate and neighbor selection (which peers hold the desired content and are likely to upload it), subscribers and upload slot selection (to whom does the peer upload?), playback policy (skip frames or wait for them?), and storage strategy (how much and what content is kept locally by the peer?).

Incentive schemes have been formulated for this specific setting (see [3] and below), but they do not include a model where peers self-maximize the benefits they gain by partic-

ipating in the system, which is what will be proposed in this thesis, using an approach based on GT.

2.2 Game Theory

GT is "the study of mathematical models of conflict and cooperation between intelligent rational decision-makers" [15]. Such a mathematical model is simply called a *game* in GT. A game comprises *players*, each of which selects from its own set of *strategies*. A *strategy profile* (or simply profile) is the result of each player choosing a strategy. Players have *preferences* over these profiles, and want to obtain the best possible profile [25]. These preferences are often modeled using a utility function, which maps profiles to values quantifying players' satisfaction.

In a given game, different concepts can be applied to obtain the profile(s) players will end up determining. One of these is the Nash Equilibrium (NE) [24]. The basic idea is that peers do not want to deviate from a given profile (change their strategy) if they cannot reach a better outcome by doing so. A profile where no player can obtain a better situation by *individually* deviating is a NE.

Quantifying Outcomes As explained in [8], the social desirability or overall efficiency of a certain profile can be quantified using different criteria, which provide a global social value for any given profile. One of these is the concept of social welfare. In this case, the sum of all players' utilities is the measure of the efficiency of a given outcome. Another possibility is the max-min function, which quantifies the outcome using the smallest individual utility among all players.

A quantifiable concept of efficiency allows for the definition of two important related measures:

- The Price of Anarchy (PoA), which is the ratio of the global value for the optimal case over the global value for the *worst* stable solution (e.g. NE);
- The Price of Stability (PoS), which analogously compares the *best* stable solution to the optimum).

Among the various types of games known in GT, this thesis makes particular use of concepts from *repeated games* and *congestion games*. These are explained in more detail in following subsections.

2.2.1 Repeated Games

Repeated games [23] are dynamic games that take place over a period of time. They may be repeated versions of static games (where all players play once and simultaneously). An important question to answer when defining a repeated game is whether and to what

extent players observe each other's moves, which determines the methods used to solve the game.

Such a game is usually represented by a tree where each non-leaf node in the tree represents a given player's turn to play the game, and leaf nodes represent final outcomes; this representation is called the extensive form. Each node in the tree corresponds to a unique succession of moves by the players leading up to that node.

2.2.2 Congestion Games

Congestion games ([8], [22]) are a type of game where a set of resources is available, and a player's strategy is the choice of a subset of those available resources. The use of each resource has a cost, which depends only on the number of players using the resource. Player's utilities include the cost of the resources they use, as well as the benefits they derive from using these resources.

A typical example of a congestion game is a network congestion game, represented by a graph (vertices are computers, edges are links). Each player has a distinct pair of computers between which they want to send data. In this case, choosing which edges (resources) to use when establishing a path between these two computers is the decision each player must make. Costs (e.g. delay) will be high if many players use the same edge.

2.3 Incentive Schemes in Streaming and Other P2P Systems

Research on incentive schemes for P2P streaming can be divided in four areas [20]: Direct & Indirect reciprocity, i.e. tit-for-tat schemes; Trust-based mechanisms; GT-based mechanisms, which are the focus in this thesis; and Tax-based mechanisms.

2.3.1 Direct & Indirect Reciprocity

The well-known case of *BitTorrent* (BT) [7] is a good example of a reciprocal mechanism. In BT, peers choose who to upload to based on how much they're downloading from them. One of the challenges is getting the peers to start cooperating (in particular when a joining peer usually has no content to offer), which is solved by periodically granting upload slots to a random peer (optimistic unchokes). This is a fairly simple algorithm, and in general it does not allow peers to maximize a well-defined payoff. Furthermore, As [11] points out, this approach is difficult to apply to live streaming, as peers who hold data far along the stream will not be interested in data from the beginning of the stream, which is what the peers interested in their data have to offer.

PSH [3] is an incentive mechanism for LiveShift that extends direct reciprocity by letting peers share records of their transactions with third parties, and redeeming their contribution in an indirect, as well as direct, way. Like direct reciprocity, this maintains a strong incentive against free-riding, but does not otherwise steer peers' decisions toward a social optimum; it also does not attempt to describe peers' incentives in detail, but rather simply assumes that some peers choose to behave well while some others choose to be freeriders.

2.3.2 Trust-based Mechanisms

These mechanisms use an evaluation of peers' contributions to determine allocation of upload bandwidth.

Scalable Peer-to-Peer Streaming for Live Entertainment Content [16] presents a protocol to transmit trust between more than two peers. Directed trusted paths are established by hops in which some peer has contributed without (yet) receiving anything in return. Trust, which is modeled as an abstract value based on peers' contributions, can be "shifted" along these paths. In a similar way to PSH, this protects against certain types of behaviors or attacks, but does not model or deal with self-maximizing peers. Additionally The proposed protocol includes a QoS-optimized overlay, which would require further changes in LiveShift.

In *EigenTrust* [12], a decentralized ranking algorithm based on the concept of transitive trust, where a peer uses its trusted neighbors' evaluations to obtain a distant peer's trust value. This decentralized algorithm is shown to converge to global and stable trust values. However, this may take a long time, which is not acceptable in a highly dynamic setting such as LiveShift. Again, the protocol targets certain behaviors in particular, but a payoff model for peers is not specified.

2.3.3 GT-based mechanisms

The following schemes are of particular interest in this thesis, as they share part of their background with the approach proposed here. However, they solve different problems, sometimes using different areas of GT.

Xiaowen C. and Kaiyong Z. [5] propose an auction-based approach, with the objective of minimizing the cost of media flow transmission.They use an acyclic mesh, constructed with respect to the order of stream playback. The media stream is separated in different flows, and must be reconstructed at its destination. The media flows are routed along the mesh according to the result of auctions. The main idea is a distributed flow auction algorithm where peers auction off and bid for flows. Bids happen sequentially; any new bid must exceed the previous bid, and the value of a bid is limited by how much the bidder values the flow. Peers check the optimality of their current flow at a given time interval to cope with system dynamics. The live-streaming specific mesh structuring and the associated routing in this approach make it inadequate for video on demand with multiple peercasters as it is currently supported by LiveShift.

[14] and [4] use evolutionary GT to find equilibria in a link sharing context, respectively to model the inter-domain relationships in a P2P streaming context. In both articles, peers choose strategies "randomly", then their choice changes depending on how successful their strategy is compared to other peers'. This is measured by a probabilistic utility, e.g. the expected value of the bandwidth available to the peer. In both of these studies, a single resource is considered and peers must choose how much they use, respectively contribute to it. It would be more difficult to apply evolutionary GT to LiveShift: in evolutionary GT, each different set of preferences has to correspond to a large population. No such thing exists in video streaming, as peers possess many different subsets of the available content, and therefore have many different sets of preferences as to what content they want to receive next. In addition, evolutionary GT requires time to converge to an equilibrium, which limits its applicability to a highly dynamic system.

EquiCast [13] is a multi-cast (i.e. tree-based) protocol for P2P streaming. It ensures cooperation between selfish peers through monitoring (where peers stop uploading to their non-contributing neighbors) and penalty mechanisms (where a "fine packet" is requested from peers who fail to contribute enough); the authors formally demonstrate that cooperation is a NE. Although EquiCast presents interesting ideas for encouraging peers to share, its reliance on a tree-based overlay makes it ill-suited for an application to LiveShift.

Multistage Congestion Games for Live Streaming ([17],[18]) proposes a static model of the P2P live streaming setting as a repeated congestion game: peers make downloading requests based on how much existing requests each source of content already has at a given step. Outcomes at each time step are described in terms of who holds what content. An optimal situation is described in that framework, and NE outcomes are compared to it. A distributed algorithm allows peers to coordinate in behaving optimally, but does not support a dynamic system.

2.3.4 Tax-based Mechanisms

Tax-based mechanisms define a tax rate, wherein a contribution from a peer only entitles this peer to a fraction of the bandwidth it contributed. The excess bandwidth thus obtained is used to reduce heterogeneity-based inequity in the system.

Chu et. al. [6] propose a taxation mechanism based on a multiple tree protocol. The flows peers send, respectively receive are determined by which trees they join as an internal, respectively a leaf node. A distributed bandwidth allocation protocol gives peers a fraction of the bandwidth they contribute. Then, the remaining bandwidth is spread uniformly between all peers (by incrementally increasing the number of trees each peer can join). An optimized tax rate leads to a marked increase in social welfare over a purely reciprocal scheme, but only in the case of heterogeneous peers. The main issue in this mechanism is an implicit assumption (or an unspecified mechanism that ensures) that peers will respect the results of the (presumably complex, not detailed in the article) distributed allocation algorithm.

Closely related work by Hu et al. [10] uses a linear tax to optimize a SVC (scalable video coding)-based P2P streaming system that provides video as layers of distinct levels

of quality. The uploaders distinguish between entitled and excess layers (this relies on knowing reliably the upload capacity of their requesters) and prioritize them accordingly. A downloader decides on the number of layers he tries to request by trying to request the next superior layer, and looking at the result of the query. Again, the incentive compatibility of the protocol is not discussed in detail.

2.3.5 Contribution

Table 2.1 sums up the characteristics of each approach reviewed here, with respect to the present thesis' goals. Those characteristics are as follows:

- type of overlay (must match or include LiveShift's mesh-pull),
- adaptability to a dynamic system (with an evolving number of peers),
- convergence speed: time needed for a mechanism to deliver full performance after a change (leave or join) in the overlay (only applicable if dynamic),
- a behavior model with peers who maximize their payoff under incentives.

As the table shows, no existing approaches completely fulfill the requirement of a dynamic system with various preferences over content such as LiveShift.

Note that the tax-based schemes, not mentioned in this table, are susceptible to manipulation from peers.

In this thesis, an approach is defined that takes the specific aspects of LiveShift into account: multiple sources of content, heterogeneous preferences over this content, a random, mesh-based overlay, and highly dynamic environment; finally, its peer behavior model allows for the maximization of individual payoff by peers.

approach	overlay type	dynamic	Convergence	Self-maximizing peers
Tit-For-Tat [7]	Any	Yes	Fast	No
PSH [3]	Any	Yes	Fast	No
Scalable P2P Streaming ... [16]	QoS-based overlay	Yes	Fast	No
EigenTrust [12]	Any	Yes	Slow	No
Auction-based [5]	Playback-order acyclic mesh	Yes	Fast	Yes
Evolutionary GT [14],[4]	Any	Yes	Slow	Yes
EquiCast [13]	Tree	Yes	Fast	Yes
Multistage Congestion Games... [17]	Any	No	N/A	Yes
Proposed approach	Any	Yes	Fast	Yes

Table 2.1: Summary table of related work

Chapter 3

Model

In the present chapter, some conventions are first introduced. Subsequently, a two-part model is presented, comprising simple assumptions about download behavior based on congestion games as in [17], as well as a more detailed description of the incentives peers face when allocating upload bandwidth. Then, a simple development shows the time optimally required to spread content of a certain length, and a metric to compare this optimum to other configurations is introduced.

3.1 Conventions and Notation

This Section discusses the structure of the model and defines elements that are used in both parts of the model. Notation is defined based on the approach in [17]. In general, the goal is to stay close to the usual conventions of game theory.

3.1.1 Model Structure

The two parts in the model represent two phases; when played subsequently (download then upload), these form a stage game. This game is repeated for a given number of steps (i.e. runs for a certain time) or until a certain condition is fulfilled (e.g. all content is completely distributed). Although it would be possible to perform both phases simultaneously, this would make each peer's decisions more complex to model, and would be less representative of the actual setting, where peers cannot know if their requests will be answered positively. Furthermore, the separation of phases allows a focus on the allocation of upload bandwidth.

Another important aspect that deviates from [17] is that the repeated game itself is not solved, only each stage is. This simplifies computation greatly by avoiding the enumeration of all possible runs of the game, with large numbers of peers and content blocks, and is justified by the lack of indirect influence any given peer's actions can have in the periods following that action: with a large number of peers and distinct blocks, it is not possible for a peer to strongly impact content distribution over the whole system.

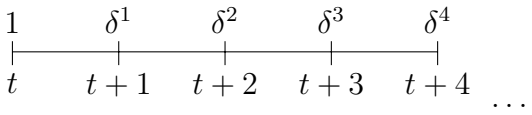

Figure 3.1: Relative time axis with valuation of blocks. t is the playback timestamp.

3.1.2 Definitions

Players (or *selfish peers*) are the peers that receive and consume stream data. They are motivated by their self interest, under certain assumptions defined later. Peercasters are not players since their actions are assumed to align with the goal of social welfare maximization. Let N be the set of players, P be the set of peercasters, and the union thereof be $M := N \cup P$. The players are selfish peers because they behave so as to maximize their self-interest, as opposed to peercasters. The set of all blocks is C.

Each selfish peer i has

- an ordered tuple of blocks $C^i =< c_1^i, ..., c_t^i, c_{t+1}^i, ... >$ that it wants to (but not necessarily will) obtain at the times corresponding to the order;
- at a given time, a set of blocks it possesses: $C_i^t = \{c \in C | i \text{ received c before time t}\}$;
- possible strategies defined differently in download and upload parts.

Peercasters only have a set of possessed blocks.

Note that the model developed here is primarily *static*: it is assumed in following calculations that no peers leave or join the system during its operation. For both download and upload behavior, the same utility function is used to reflect the relative desirability of pieces for a selfish peer i. This function uses a discounting factor δ for blocks that are not immediately needed. δ is positive and smaller than 1; it stands for the selfish peers' valuation of future blocks that cannot be locally played yet (which is high if δ is high). All earlier blocks are valued zero. A visualization of that utility function is shown in Figure 3.1; e.g. if peer i is waiting to play the block with timestamp t, i would give block c_t^i a value of 1, and block c_{t+2}^i a value of δ^2.

Formally:

Definition 1 (Utility of single blocks)**.** Given time v and $c_v^i \notin C_i^t$,

$$u_i^t(c_v^i) := \begin{cases} 1 & \text{if } v = t \\ \delta^{v-t} & \text{if } v > t \end{cases}$$

Additionally, $u_i^t(c) = 0 \ \forall c \notin (C^i \setminus C_i^t)$, and $u_i^t(c_v^i) = 0 \ \forall v < t$

Observe that the utility value depends on when and whether the selfish peer can use the block, assuming it gets the block at the present time. In term of priority given to blocks, this is very similar to existing LiveShift policy. However, it adds a quantification of the utility each block potentially gives to a peer, allowing the evaluation of a block's, and therefore a peer's, potential popularity among other peers (see Subsection 3.3.2).

3.2 Download Behavior

In this part of the model, the upload decisions of players are not analyzed explicitly; the focus is on the choice that players make when sending requests to download a specific block. Following [17], a congestion game model of the P2P streaming setting is used. Congestion games provide an adequate and well-known framework for download behavior. One simplification made here is that players select desired blocks independently of the resources that hold these; the choice of a resource is subsequent and subordinated to block selection. This implies the assumption that selfish peers will not *proactively* skip blocks to avoid congestion; following current LiveShift policies [9], they will skip them when a sufficient amount of more timely blocks become available.

Let G be a repeated congestion game, with the set of resources M and players N.

Definition 2 (Download strategy)**.** Assuming selfish peers are limited to one request per turn or stage, a strategy s_i^t (for turn t) is defined as

– the choice of a resource $j \in M$,
– a block c to request from j,

for each stage that the game runs. If more than one request is possible, a strategy is a set of (j, c) pairs.

Each resource j has a cost function $\gamma_j(n_j^t)$, where n_j^t is the number of requests directed at resource j at time t, quantifying the load of that resource at that time. γ_j must be strictly monotone in n_j^t:

$$n_j' > n_j^t \Rightarrow \gamma_j(n_j') > \gamma_j(n_j^t) \forall j$$

This means that, given a desired block, a selfish peer will always request from the least busy resource that is able to provide that block. Therefore, as long as there are more requests for a given block than there are copies available, a resource which has a copy of that block will always get at least one request. For this thesis' purpose, $\gamma_j(n_j^t) := n_j^t$.

Before it can select a resource to request from, a player must decide which block(s) it is going to request. Given a certain number l of requests per turn, it establishes a ranking of the most desired pieces according to u_i^t (Definition 1) and selects the top l pieces. It then sends requests for each of these so as to minimize

$$\pi_i^t := \sum_{(j,c) \in s_i^t} \gamma_j(n_j^t)$$

, its cost function.

Note that the congestion information is the number of requests received; this is a concern for implementation, as transmitting that information reliably and without manipulation is not trivial in a selfish setting. However, peers wanting to use all their bandwidth have an incentive to properly report their (lack of) congestion: if they pretend to have higher-than-actual congestion, they will not receive as many requests and may therefore not upload as much, thereby hurting them in a tit-for-tat environment. As for purposefully

giving a lower value of congestion, this will mostly be of interest if there are not enough requests for peers to use all of their upload bandwidth, meaning that upload bandwidth is plentiful; as such, inefficiencies due to manipulation will not matter at all. This last argument can also be made regarding the upload rate.

3.3 Upload Behavior

Upload bandwidth is a scarce resource. As such, it is important to know how it should be allocated, as well as how it actually is allocated. In this thesis, it is assumed that there is no free-riding by selfish peers, i.e. all of their upload bandwidth is made available. This is derived from the hypothesis that a scheme to counter free-riding is already present in the system (such as PSH [3] in LiveShift).

3.3.1 Optimal Behavior

Following [18], the optimal distribution of content requires copies to be uploaded in a specific way. In this ideal situation, players are not behaving selfishly, but rather behave so as to maximize social welfare. Concretely, this means that until a block has been fully distributed, a player that holds that block must

1. upload it with its full upload capacity,
2. and only to players who do not already have a block that they can upload in the next stage.

The peercaster must also abide by the second rule, but it always uploads the latest piece in the stream. If these rules are not respected, some players will end up holding more than one block that is needed by others, thereby dividing their upload between those blocks and missing opportunities to upload copies. Note that these rules of behavior imply that peers must upload first to those who do not have any blocks, or only undesired ones. An example of optimal behavior is given in Figure 3.2, which is a representation similar to what is shown in [18]. In the depicted situation, analogous to [18], all peers have an upload capacity of 1. Observe that at any point in the process, a selfish peer never has to choose which block it should upload, as it only ever possesses one desired block. Such a configuration requires peers to be highly coordinated; as such, it can only be implemented in a deterministic fashion, where all peers and their desired content are known in advance. In the following, an approach is proposed that attempts to be as close as possible to this optimum situation, while remaining compatible with a dynamic, unpredictable, distributed system (Subsection 3.3.2, “Peercaster Behavior and Well-Behaved Peers”).

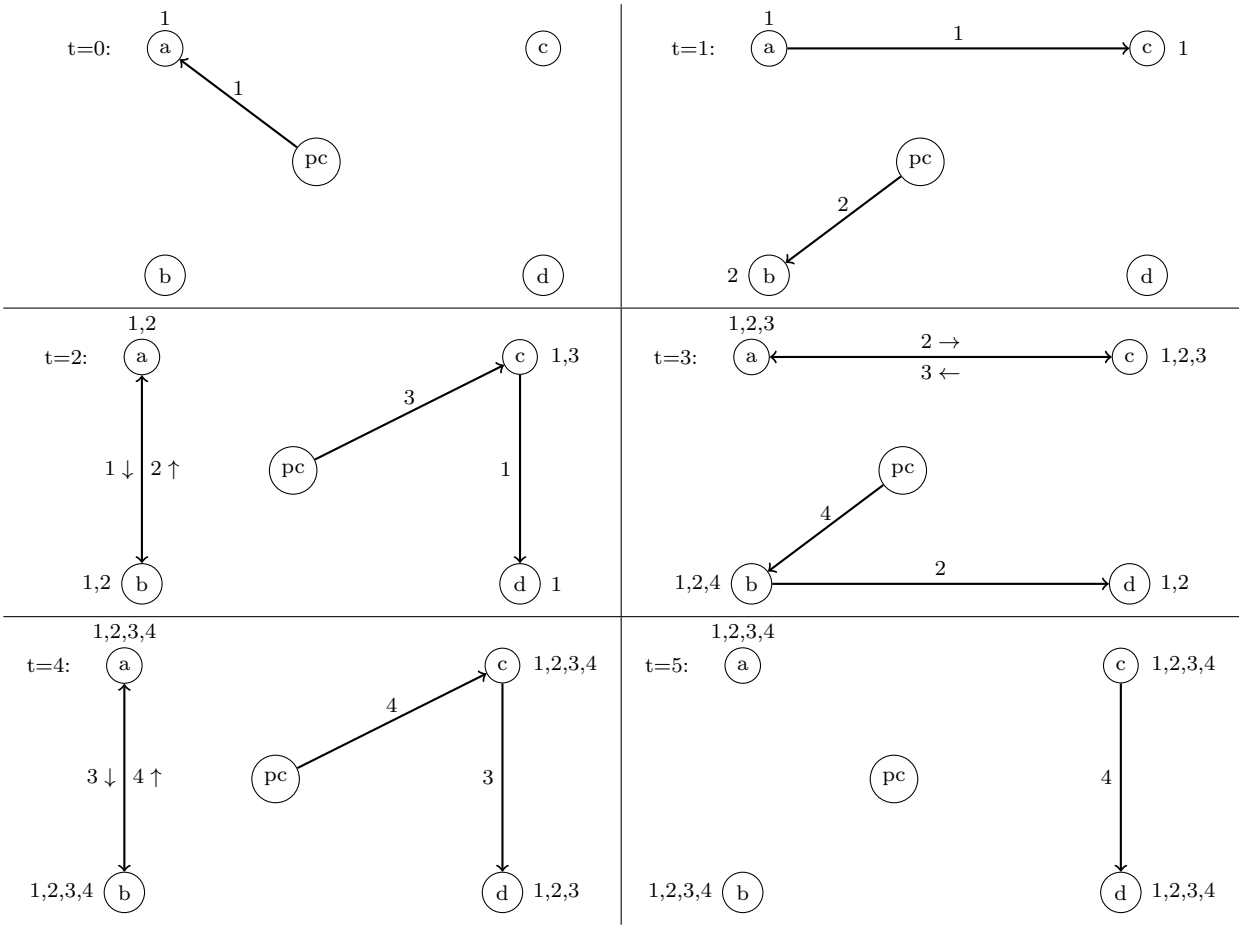

Figure 3.2: Graphs of upload operations for successive steps.

3.3.2 Incentives and Strategies

The upload strategy of a selfish peer is a selection from the set of requests addressed to it in the download phase. The number of selected requests must be equal to its upload rate (in streams) μ_i.

Definition 3 (Upload Strategies)**.** Recall s_i^t as in Definition 2. The set of possible upload strategies for player i is

$$A_i^t = \{S = \{(j,c) \in |N| \times |C| \text{ s.t. } (i,c) \in s_j^t \text{ and } c \in C_i^t\} \text{ s.t. } |S| \leq \mu_i\}$$

When uploading, selfish peers try to maximize the return they may get from others as a consequence of their actions. Assuming that incentives against free-riding (such as TFT or PSH) are in place, i.e. a player has to upload, it will prefer uploading to those candidates who hold content it is interested in. Observe that this may lead to behavior that deviates from the optimum.

Let $X_i^j(t) := (C^i \setminus C_i^t) \cap C_j^t$. Then, recalling u_i^t as in Definition 1, the expected payoff for uploading to a given selfish peer, without taking actual reciprocation into account, is

$$\Phi_i^t(j) := \sum_{c^u \in X_i^j(t)} u_i^t(c^u)$$

Popularity

Additionally, in order to adapt the idea of a "peer that holds only unwanted blocks" outlined in Subsection 3.3.1, the overall popularity of the player's content must be taken into account; it is a sign of future congestion and reduces the probability of future reciprocation. Here, the popularity of a single block is characterized as the sum of all peers' utility values for that block. This is then divided by the number of copies that are potentially available, e.g. the sum of upload capacities of all peers who possess that block.

In turn, the popularity of a player k's content is a sum of all the popularity values of the blocks k possesses, and normalized by k's upload rate. It is characterized by the coefficient

$$\omega_k := \frac{1}{\mu_k} \sum_{c^u \in C_k^t} \frac{\sum_{j \in N} u_j^t(c^u)}{\sum_{i \in M | c^u \in C_i} \mu_i}$$

This coefficient is computed in such a way that the information needed is who possesses which block and what their upload rate is. This assumes that the block utility function (Definition 1) has the same δ values for all peers. Otherwise, these values need to be known as well. The advantage of using block information is that the LiveShift system already maintains segment information in a decentralized way, which limits how far peers can manipulate information about blocks they own. Direct information about how many requests

a given player received may be more easily manipulated and will change more quickly. As for manipulating that information, lying to the peercaster about one's possessed content in order to gain an advantage would mean pretending having fewer blocks than actually possessed. To the extent that this impacted the distributed segment information, this would mean a loss of attractivity towards some selfish peers, i.e. a disincentive to lie in such a manner. Therefore, even in an environment where peers misrepresent their upload rate, the index may still incorporate some valid information.

The computation of the popularity index presents a tradeoff in terms of computing efficiency vs. reliability of information and network efficiency. It is quicker for each peer to compute its own popularity value and then send the result, rather than for all the peers to compute all values locally, but the former alternative may cause congestion and would let the peers manipulate their popularity values.

Payoff Function

As such, the popularity-normalized partial payoff is defined as:

$$\overline{\Phi_i^t}(j) := \frac{\Phi_i^t(j)}{\omega_j}$$

, where $i, j \in N$. Finally, priority is given when there is reciprocation within the same stage (i.e. two players have made requests to each other, and granted each other's request). The actual payoff function takes this into account:

Definition 4 (Payoff Function)**.** Given a strategy $\sigma_i^t \in A_i^t$, the payoff function is

$$\Pi_i^t(\sigma_i^t) := \sum_{(j,c)\in\sigma_i^t} (\overline{\Phi_i^t}(j) + \sum_{k|(i,k)\in\sigma_j^t} u_i^t(k))$$

Selfish peers who sent a request to i are ranked according to their contribution to that sum. This defines a best response for each player given the other player's strategies. Iteration will usually be necessary to arrive at an equilibrium. The payoff function can be seen as a heuristic that selfish peers use to select which requests to grant. This heuristic is needed because the task of solving the stage game, having too large memory and computing requirements, is not attempted. Since future behavior is hard to predict, the payoff function puts more weight on the actual contributions performed at the present time. Past behavior is not taken into account because it is assumed that a scheme to prevent free-riding is active, and the system is highly dynamic (e.g. peers change channels and "diverge" in preferences, a lost connection causes a previously good resource to stop being up-to-date, etc.).

Peercaster Behavior and Well-Behaved Peers

In a realistic setting where selfish peers act autonomously, their behavior can be influenced by the peercaster. The existing peercaster behavior is to select players to upload to based

on their advertised upload bandwidth as well as the amount of data they successfully downloaded from the peercaster in the past [9].

However, here we assume that the peercaster's goal is to bring the selfish peers' behavior as close to the optimal behavior as possible. This approach is valid in cases where the peercaster is interested in its own content and the distribution of it. It does not apply if the content is being offered by the peercaster in exchange for access to other content (analogous to certain file-sharing scenarios). The basic way the peercaster can maximize social welfare is to stick to its behavior as described in Subsection 3.3.1: always upload the latest block in the stream to the least popular peer, taking upload speed into account. Thus, all selfish peers will quickly acquire interesting fragments, creating mutual incentives for sharing among themselves. To rank the peers, the popularity coefficient ω_k is used, ranking peers in increasing order. Note that to be maximally effective, this type of behavior requires that peercasters be able to upload to players that did not necessarily send them a request, as those with the least amount of popular blocks may be "late" in the stream, and therefore not immediately interested in the latest block. Because of its focus on low-popularity peers, the approach is expected to be particularly helpful to peers who tend to be "left behind" in the overlay because of a lack of interesting blocks to share, ending up with very few acquired blocks in comparison with most other peers.

Well-behaved peers have the same behavior as the peercaster, except they always grant the least popular peer its request, instead of trying to grant the latest block received. They do so because in a time-shifted system, the latest block may not always be the limiting factor (at least once it has been released by the peercaster). In a dynamic system, all peers being well-behaved should produce results closest to the optimum.

3.4 Optimal Configuration

In this Section, a minimal duration for distributing a stream of a certain length is computed. In order to compare other configurations of the system with this optimal one, a metric is proposed. It uses a direct measurement of the system's performance, avoiding dependency on hypotheses made earlier concerning the selfish peers' preferences.

Let $\bar{\mu}$ be the average upload rate and ρ the initial number of copies (i.e. the peercaster's upload rate). Assuming upload behavior is optimal as defined above, a block, once released by the peercaster, will on average see its number of copies multiplied by $\bar{\mu} + 1$ each turn (incl. already existing copies). Therefore, the expected total number of copies of a block at time t is given by

$$\rho(\bar{\mu} + 1)^{t-1}$$

(It is *t-1* since in the first period, the upload rate is ρ). Knowing this, the time needed for a piece to be distributed among all players can be computed (assuming the peercaster does not contribute to the upload after releasing the block) by solving the following equation:

$$\rho(\bar{\mu} + 1)^{t-1} = n \Leftrightarrow$$

$$t - 1 = log_{\bar{\mu}+1}(n) - log_{\bar{\mu}+1}(\rho) \Leftrightarrow$$

$$t = log_{\bar{\mu}+1}(n) - log_{\bar{\mu}+1}(\rho) + 1$$

Since t only makes sense as an integer, use the ceiling value:

$$\lceil t \rceil = \lceil log_{\bar{\mu}+1}(n) - log_{\bar{\mu}+1}(\rho) \rceil + 1$$

Finally, let there be T blocks of content to distribute. Then, the expected time when the distribution of the last block is completed is

$$T + \lceil t \rceil = T + \lceil log_{\bar{\mu}+1}(n) - log_{\bar{\mu}+1}(\rho) \rceil + 1$$

Note that different blocks will take varying amounts of time to spread. This depends on which peers receive a block last (and therefore cannot contribute upload bandwidth to the distribution of that particular block). On average, however, the result above is the duration resulting from optimal behavior.

When multiple peercasters are involved, no general formula can be computed, as the necessary time will depend on peers' preferences between the different streams. If all peers are only interested in a unique stream (worst case), the values are the same as above; if peers are spread evenly among the different peercasters (best case), then the number of peers n must be divided by the number of peercasters. The cases in between those two extremes would need to be computed on the basis of the distribution of peer's preferences, taking into account if relevant any dependencies between preference and upload capacity; this is outside the scope of this thesis. The possibility of time shifting also changes the time required to distribute components: for peers that are (even slightly) behind the real time stream in terms of what they want to play, then it is possible (in a multi-peercaster environment) for these peers to receive T desired pieces before time T is reached. This can result in the total time necessary being less than T itself.

3.4.1 Performance Metric

In order to measure the overall performance of the system, a metric is needed. This performance metric needs to take the overall level of satisfaction of peers into account. Although assumptions are made elsewhere in this thesis about the factors that determine that satisfaction, the chosen metric should be largely independent from these assumptions. Thus, a criteria that is based solely on the traffic going through the system over a certain period of time is desired.

The metric is defined as a number of copies distributed over a certain time, with the optimal value defined by the example above. In general, the metric is

$$K := \frac{\sum_{i \in N} |C_i^\theta|}{\theta}$$

Where θ is the time limit or the time when distribution is complete. The theoretical optimal value would therefore be:

$$K^* = \frac{T \cdot n}{T + \lceil log_{\bar{\mu}+1}(n) - log_{\bar{\mu}+1}(\rho) \rceil + 1}$$

Although the metric is simple and assumption independent, it has the disadvantage of not directly taking certain measurements of quality into account, such as lag or number of frames skipped (see e.g. [9]). It only shows how much content was transmitted in how long, which is indirectly related to skipped or stalled blocks.

Despite this shortcoming, the metric is appropriate for the present approach, which is social welfare-based (i.e. the optimum is calculated based on a sum of individual values). Measurements of lag or skipped frames make little sense when combined in a sum or average; a min-max approach would be more appropriate for these.

Having presented a model and discussed its various advantages and disadvantages, in general as well as with regards to an implementation in the LiveShift system, a simulation can now be presented and experiments be run to test that model.

Chapter 4

Results and Evaluation

In this chapter, the simulation is first presented, as well as the various experiments it is used to run. Then, the results of these experiments are shown and discussed.

4.1 Simulation

A simulation is created based on the model, in order to show some of its properties and compare it to the upload and download policies originally used in LiveShift. It allows for running more experiments and tweaking the model more quickly than an actual implementation would. Also, its result may be more of more general scope as it does not need to strictly implement all the particularities of the LiveShift system. This section details its implementation and the settings with which it is run.

4.1.1 Implementation

The simulation is implemented using a general-purpose platform (here Java 1.6). It comprises a basic model of the setting, including blocks, selfish peers, peercasters, and the requests they receive and grant each other; the congestion and upload allocation games and their rules; and a module that outputs the result. The present section will focus on the implementation of the game logic.

Both game implementations define a process, called a dynamic, through which a selfish peer changes its strategy in response to the current state of the game if there is a possibility for it to increase its utility. To reach an equilibrium, this process is repeated until no selfish peer changes its strategy anymore. Utility functions evaluating both blocks and other peers also have to be implemented for the games implementing the proposed approach. For performance purposes, utility values are cached until they become invalid.

Player preferences

Player's preferences in terms of content are defined randomly at the beginning of the game. For each player, an ordered set of desired blocks is computed as follows:

1. A peercaster (channel) is chosen at random; the first desired block is this peercaster's first released content block.
2. In most cases, the next block from the same channel is then selected. However, there is a one percent chance to switch to a different channel and playback time. The new playback time is uniformly distributed between one and the current time.

Congestion Game

In the congestion game, the dynamic a selfish peer follows has the following steps:

- The peer determines its order of preference for blocks.
- Then, for each block:
  - sort the possessing peers according to the valuation function for peers (see Section 3.3.2);
  - send requests for the block to each peer in this order, until there are no more peers to request the block from, or the maximal number of requests is reached and all the peers from whom the block already has been requested have a lower value of $\gamma_j(n_j^t)$ than the current peer, i.e. are better candidates;
  - if the maximal number of requests has been reached, a request must be withdrawn from the least attractive peer every time a new request is sent.

This dynamic will often have to repeated, as one peer sending a request will influence the preferences of other peers, sometimes prompting changes in strategy.

Upload Allocation

In the upload allocation game, both the peercaster and the selfish peers must make a decision; in well-behaved cases, with the proposed approach, the selfish peer is replaced by a well-behaved peer, which has a distinct decision-making process.

A peercaster starts by sorting all the selfish peers in increasing values of ω_i. Then, going through that sorted list, the peercaster sends the latest block to each peer if they are interested in it, even at a much later time. The peercaster keeps going through the list until it exhausts its upload or no more selfish peers are interested in the latest block. Then, if it has not exhausted its upload rate, the peercaster goes through the sorted list of selfish peers once again, this time granting all requests to each selfish peer until its upload is completely used or the end of the list is reached.

A selfish peer sorts its requesting selfish peers i in decreasing according to the value of $\overline{\Phi_i^t}(j) + \sum_{k|(i,k)\in\sigma_j^t} u_i^t(k)$, which is the contribution of a peer to the selfish peer's payoff function (see Subsection 3.3.2). Until its upload rate is reached or it has granted all

block requests, the selfish peer grants all requests for each peer in this order. If any of the selected peers and blocks are different than in the previous iteration, then another iteration of the upload allocation game must be executed.

A well-behaved peer will proceed as in the second part of the peercaster's behavior: first sorting all requesting peers by value of ω_i, then granting each requesting peer all of its requests, until the upload rate is exhausted.

Existing LiveShift policies

LiveShift policies are implemented in a simplified way. Some steps in the protocol are omitted. In the well-behaved case (see Section 4.1.2), all peers behave as described in this Subsection. In the selfish setting (also in Section 4.1.2), only the peercaster behaves as in the existing LiveShift policy; the other peers use selfish behavior as defined in Subsections 4.1.1 and 4.1.1.

Download Peers have a group of candidates constituted randomly at the beginning, where, in accordance with the LiveShift protocol, members that match the following criteria are removed:

– peers that send more than five requests without uploading in return;
– peers that only have blocks older than the playback time minus eight.

When deciding who to download from, requests for any of the desired blocks (any not yet acquired after the last played block) are made in chronological order of blocks, to any candidate who has them. There is no limit to the number of requests made.

PeerSuggestion messages are simulated in the following way: if, for a given peer, a desired block is not possessed by any of the existing candidates and the peer possesses the previous block, a new peer possessing the missing block is added. Peers who have sent the most blocks recently will be added first. If, after this operation, the size of the candidate group exceeds 40, the candidate who sent the least blocks recently is removed. Senders of block requests that the peer receives are also added if they have interesting blocks. This happens if the peer is not receiving blocks fast enough (i.e. if its lag increased in the previous step) and as long as there are less than 40 candidates. In this case, requesters are added according to the block they possess (in chronological order from the playback time) and secondarily according to how many block they recently sent to the peer.

Upload Both normal peers and peercasters sort requesters by upload rate, secondarily by blocks sent to the requester in the past. For each requesting peer, the peercaster grants the latest piece if it is desired by the peer, otherwise it grants the first request it got from that peer on this turn; it does so until its bandwidth is exhausted. Normal peers simply grant the first request received from each requester, also until exhausting their bandwidth.

4.1.2 Experiments

The base setting for all the experiments has the same parameters as scenario three in the LiveShift TR [9], except all the peers are present at the beginning (instead of being added progressively). This means that in the standard setting, there are 6 peercasters, 15 high upload (HU) peers, and 120 low upload (LU) peers. Both peercasters and HU peers have upload capacity of five times the bandwidth necessary to stream the content to one target. LU peers have only half the stream bandwidth in terms of upload. The amount of content to distribute is 500 (one-second) blocks. This setting is run with changes in player behavior (well-behaved vs. selfish) and with different levels of overlay stability (changed by the addition of flash crowds).

Player behavior

In the experiments, players (in terms of their upload) can be selfish, use the well-behaved behavior defined in Subsection 3.3.2 with some modification, or use the existing LiveShift policy. As far as download behavior is concerned, congestion game and existing LiveShift policy are the only two choices.

The ideal case compares LiveShift policies (for all aspects of player and peercaster behavior) with well-behaved players and peercasters, using the congestion game for download behavior; the realistic case compares the LiveShift policies and well-behaved upload (3.3.2) for the peercaster, who faces selfish players (behaving according to congestion games and the selfish upload behavior defined in Subsection 3.3.2).

Overlay stability

In the standard scenario, all peers are in the system from the beginning. To test for the different approaches' resilience to perturbation, all experiments are then also ran with flash crowds. The flash crowds have the same proportions of upload bandwidths as the base system (including the peercasters). This ensures that the system's amount of upload bandwidth per player remains the same. It is also better than simply having a flash crowd that is homogeneous with respect to bandwidth, as it preserves variability as defined by the standard deviation over the population of peers.

Flash crowds consist of two events:

- at $t = 100$, 50 peers are added. All of these peers are interested in current content of a single peercaster. These peers stay in the overlay (without changing channels) until $t = 200$, and then disconnect.
- at $t = 150$, 75 peers are added. All of these peers are interested in current content of a single peercaster, distinct from the one in the previous event. These peers stay in the overlay (without changing channels) until $t = 250$, and then disconnect.

To summarize, eight experiments are run in total, varying in policies used (LiveShift vs. the proposed approach), player behavior (selfish vs. well-behaved), and overlay stability (stable vs flash crowds). In the next chapter, the results from these experiments are presented and evaluated.

4.2 Results

Selected results from the experiments ran in the simulation are presented and discussed here. The cases with well-behaved, respectively selfish peers are distinguished. In each of these, results with a stable peer population and with a changing one (flash crowds) are presented.

The metric results are shown, but they do not provide a complete picture of the quality of experience (see [9]): as the metric K is simply the number of distributed copies (of any block) divided by time, it does not reflect the order in which blocks are received. Therefore, and also because the number of distributed blocks is not used in other evaluations of LiveShift, results on playback lag over the peer population are provided in the following form:

- the metric results at $t = 500$, the time at which no more new blocks are produced by the peercaster,
- a graph of the median and 95th percentile over time,
- and a graph of the cumulative distribution of lag at the time ($t = 500$) when the streamed content ends.

Values at $t = 500$ is of particular interest, as they gives a good idea of the performance over time without being influenced by the lack of new blocks to distribute (which means lower performance will be less apparent, as the bottleneck becomes the number of distinct blocks in the system as opposed to the number of copies). All graphs compare the current LiveShift policies with the proposed approach, for each graph under different conditions. The data represented are median values taken over 50 runs of each experiment. Higher values of lag are observed overall, as the simulation does not implement a reaction to failed playback as defined in [9].

4.2.1 Well-behaved case

Results from the experiments where peers are well-behaved are presented here.

Stable Scenario

The metric results are the following: at $t = 500$, the well-behaved approach has a value of $K = 135$; the LiveShift policies have $K = 130.7$. Although this shows a slightly better ability of the proposed approach to distribute blocks more quickly (which is expected), the difference is relatively weak.

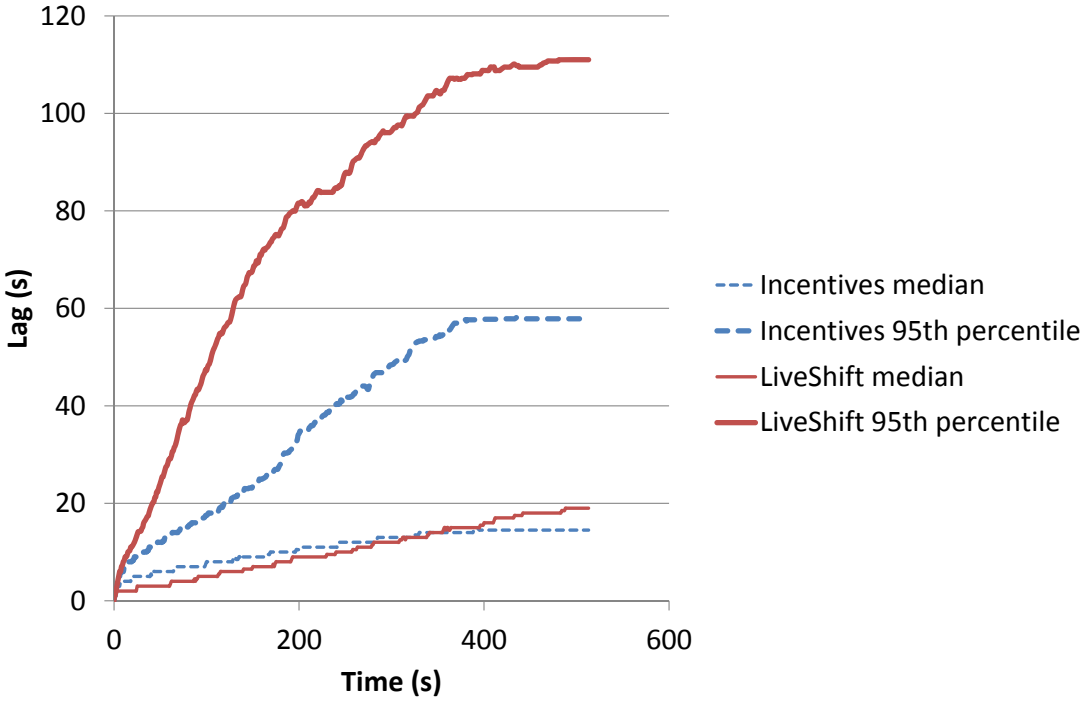

Figure 4.1: 95th percentile and median lag in the well-behaved, stable setting.

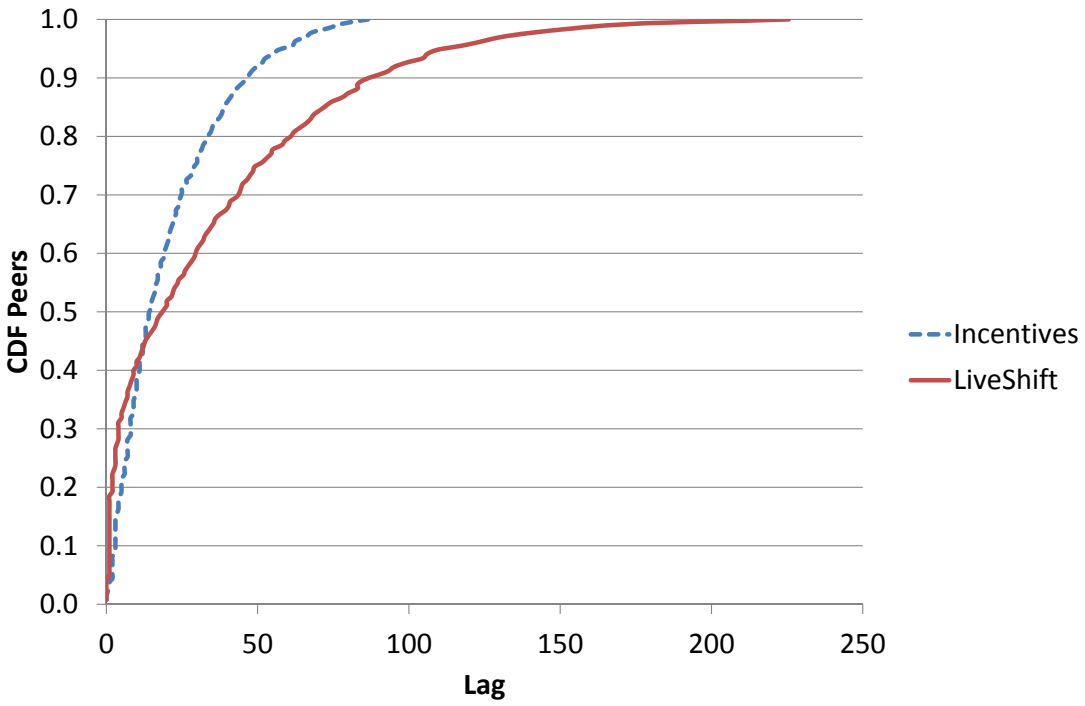

Figure 4.2: Cumulative distribution of lag at the end of the experiment with well-behaved, stable peers

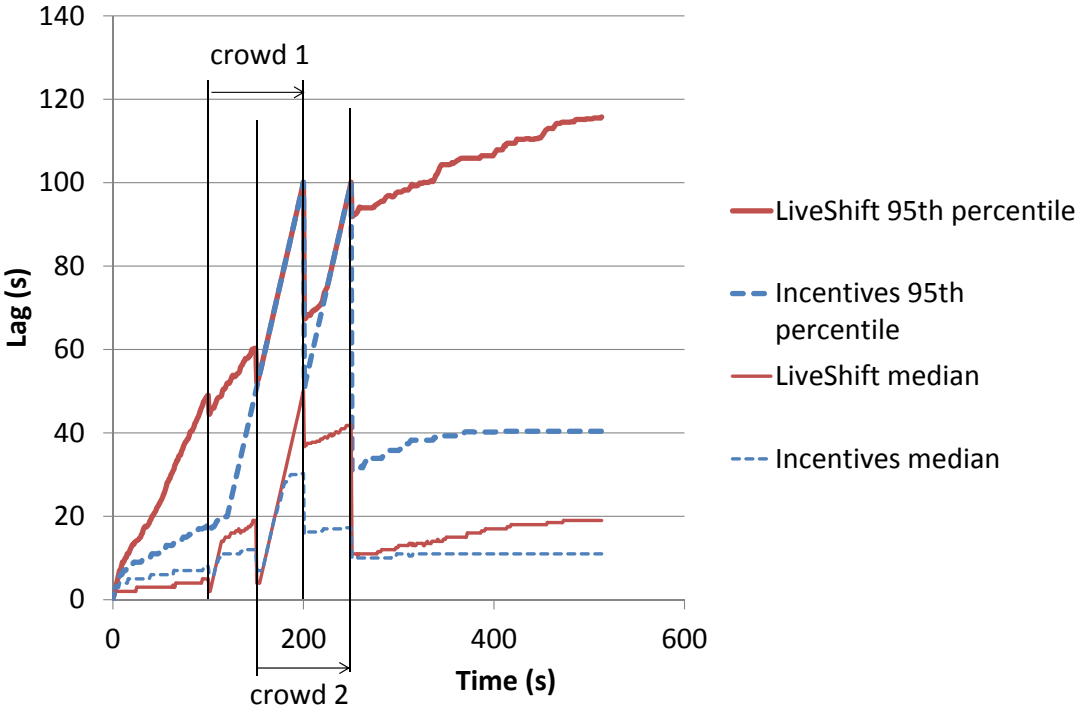

Figure 4.3: 95th percentile and median lag over time in the well-behaved setting, with flash crowds.

In terms of lag, the differences observed between LiveShift policies and this thesis's proposal are in favor of the latter. As Figures 4.1 and 4.2 show, LiveShift policies perform worse both in terms of extreme values and in general for the majority of peers. In that last case, however, the difference is very limited. For the 40% peers with the lowest lag, this difference is even reversed in favor of the current LiveShift policies. These variations exist because the proposed approach will distribute blocks in a more even way than LiveShift, limiting extreme values of lag. On the other hand, peers will upload to more distinct requesters, and will switch who they upload to more often. This will result in more interruption of the stream for peers who are usually closer to the peercaster in the overlay and who would otherwise enjoy smoother playback. This effect is more visible in the beginning of the streaming session (see Figure 4.1), as the indirect advantage – more overall bandwidth utilization – created by spreading blocks more evenly is felt later on in the session and balances that effect somewhat.

Flash Crowd Scenario

As far as lag is concerned, a clear difference in results is observed in Figures 4.3 and 4.4: peers with median values of lag and peers with very high values of lag are better off. The size of the difference is much greater than in the stable scenario; this is explained by LiveShift's emphasis on the maintenance of stability in the overlay, which implies it is slow to take advantage of additional peers (in terms of utilizing their bandwidth), resulting in higher lag after a certain time. On the other hand, the approach proposed in this thesis always gives priority to peers who do not have any content yet, resulting in

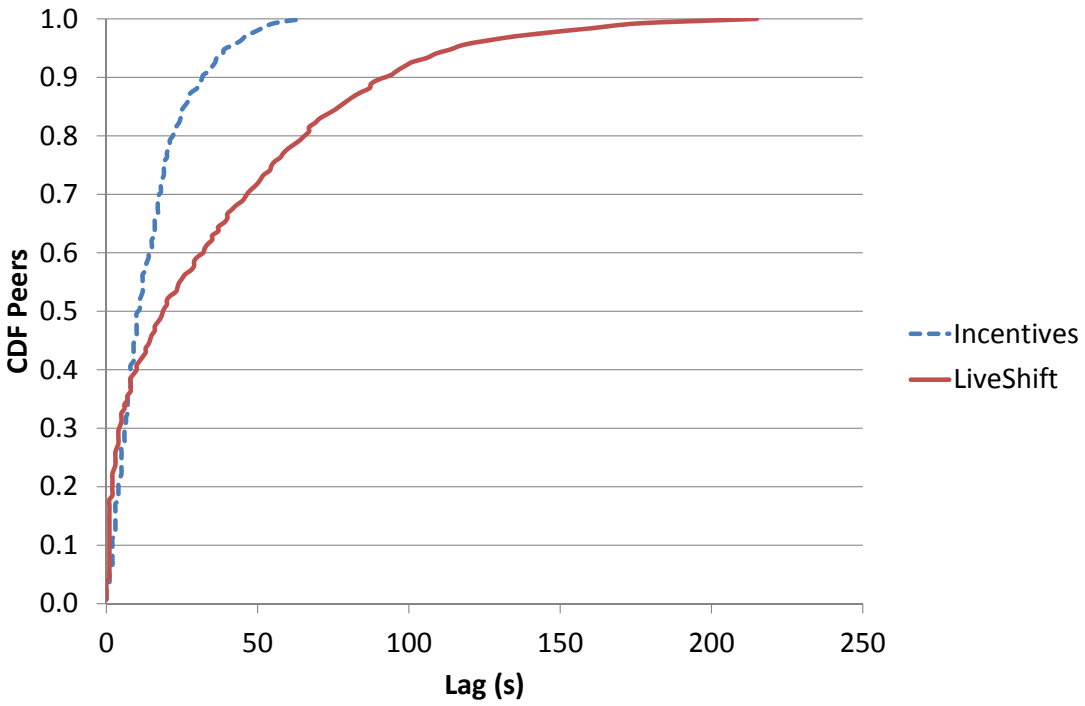

Figure 4.4: Cumulative Distribution of lag at $t = 500$ in the well-behaved setting, with flash crowds

more bandwidth usage in this case. However, peers below the 40th percentile are again slightly worse in terms of lag; this is expected as the existing LiveShift approach will consistently favor peers that have successfully received more blocks in the past, whereas the proposed approach does not.

The spikes in Figure 4.3 are caused by peers massively joining and leaving the overlay in flash crowd events. At $t = 100$ and 150, a flash crowd joins the overlay: this causes an immediate drop, as peers join the overlay with a lag of zero, pushing all percentiles down (this is more visible with the second event, as more high values have had time to accumulate); and then a progressive spike (sometimes slightly lagged) in all curves. At the start of an event, the reaction in both approaches is similar: the spikes are of the same height in both 95th percentile curves, and the median levels off much more quickly for the proposed approach, which is expected given the focus on spreading blocks evenly. When the flash crowds depart, at $t = 200$, respectively 250, a very sharp drop is again observed in both approaches, as many members of the flash crowds have developed high values of lag. The proposed approach distinguishes itself in this case, benefiting more from the flash crowds' departure in the case of extreme values, especially in the last event. This shows that the underlying block distribution was better during the flash crowd, even though it could not be seen before the flash crowd left, as the system was starved of usable bandwidth during that time. Combined with the behavior of the median when flash crowds move in, this provides strong evidence that avoiding congestion and aggressively enabling the use of bandwidth can strongly improve performance.

In terms of the metric, at $t = 500$, the well-behaved approach has a value of $K = 135$; the LiveShift policies have $K = 130.9$. The difference is almost the same as in the stable

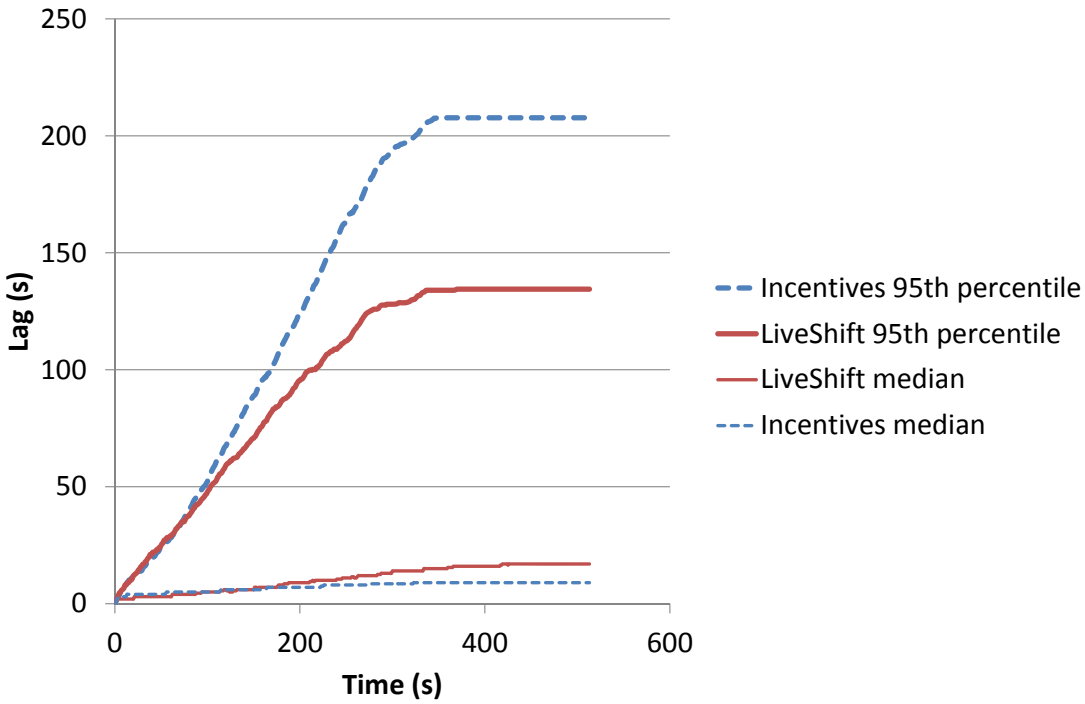

Figure 4.5: 95th percentile and median lag over time in the selfish, stable setting.

scenario; this would seem to indicate an absence of impact flash crowds. However, the lag results show that despite a similar number of copies distributed, the exact timing of the distribution was quite different in the two approaches.

4.2.2 Selfish case

Stable Scenario

At $t = 500$, both approaches have a value of $K = 135$. No useful distinction can be drawn between the approaches, although this indicates that the LiveShift policies, once applied only by the peercaster, performed better.

The lag differences are not uniformly in favor of one approach. As Figure 4.5 shows, the incentives reduce lag for median peers, but increase extreme values of lag. In this case, the peercasters' behavior does not help the peers that are very late in the stream, as it mostly shares the latest block. Very late peers may not have any interest in this block, and even those interested in that block may not hold content a late peer has immediate interest in.

The upside to this situation is seen in the median: late peers have no other popular content, so they distribute the few blocks they get from the peercaster very efficiently. This is also seen in Figure 4.6, where over 70% of the peers have values of lag that are lower or nearly identical; although it must be noted that the average values are clearly higher overall for the proposed approach : 46.6 seconds, compared to 38.5 seconds for LiveShift. This

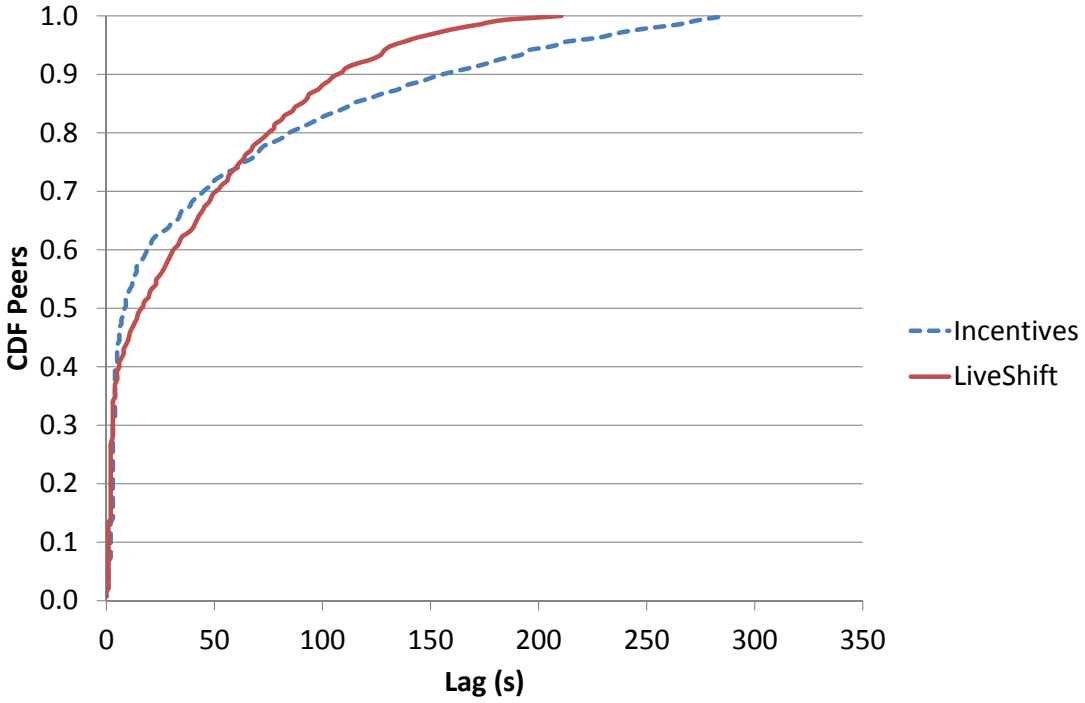

Figure 4.6: Cumulative distribution of lag at the end of the experiment with selfish, stable peers

is partly an effect of extreme values, but clearly hints that the two approaches perform similarly in this case. However, aggregates such as sums or averages make relatively little sense when applied to values whose effect on perceived performance is neither linear nor applies identically to anyone.

Dynamic Scenario

At $t = 500$, both approaches again have a value of $K = 135$.

As in the well-behaved case, the proposed approach performs better in the flash crowd scenario than in the stable one; although as Figure 4.7 shows, that does not allow it to overtake LiveShift in terms of extreme values of lag, the difference is significantly reduced. This is also observed in the cumulative distribution (Figure 4.8), where around 50% of the peers are better off (between the 40th and 90th percentile), and an additional 40% see next to no performance difference. These results are explained by the combination of positive impacts on the proposed approach in the flash crowd scenario, i.e. the approach's ability to quickly leverage bandwidth from newly arrived peers, on the one hand; and the negative impacts from the selfish scenario, namely the inadequacy of the peercaster behavior in truly helping peers with extreme values of lag, on the other hand. Like in the flash-crowd scenario in the well-behaved case, spikes and drops are noticeable at / after join and leave events. As far as extreme values of lag are concerned, the spikes and drops are not as noticeable, because selfish behavior creates more extreme values of lag by isolating certain peers, and because neither approach can take substantial advantage

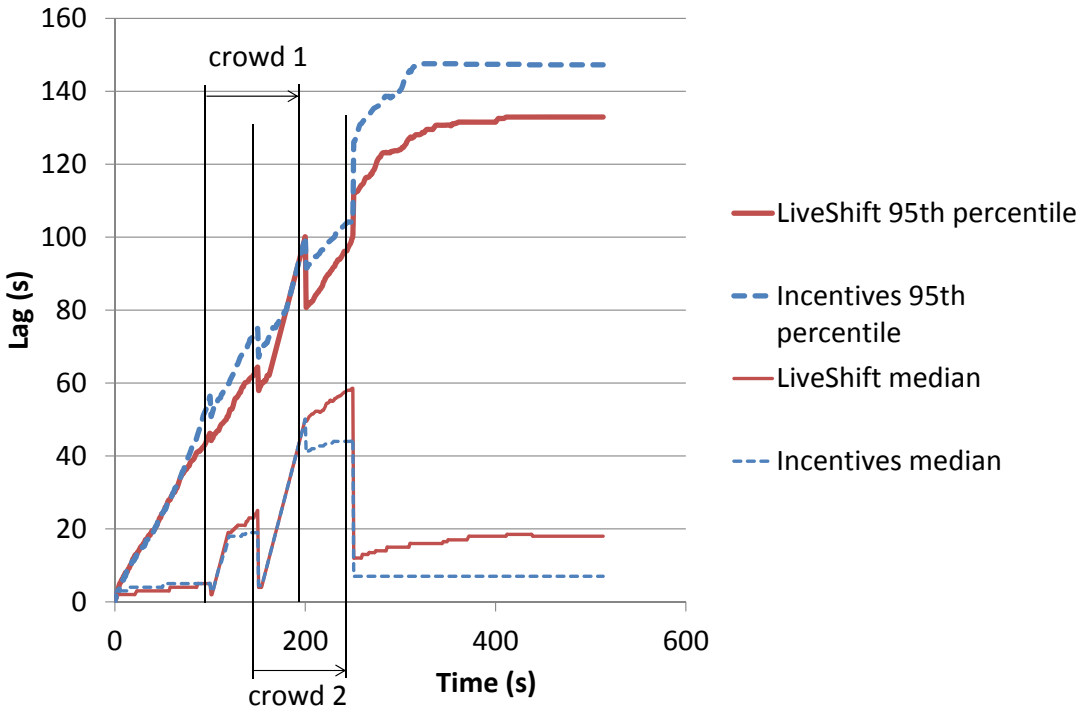

Figure 4.7: 95th percentile and median lag over time in the selfish setting with flash crowds

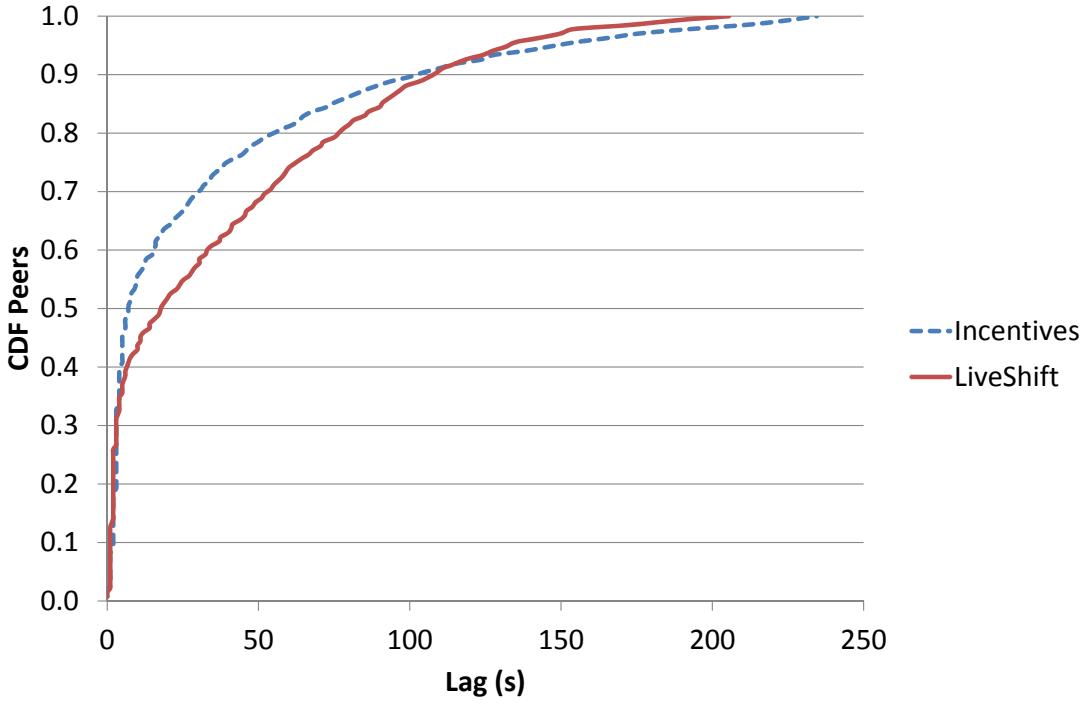

Figure 4.8: Cumulative distribution of lag at $t = 500$ in the selfish experiment with with flash crowds

of the flash crowds' bandwidth to limit extreme values when limited to determining the peercaster's behavior. The median is however still as sensitive to the flash crowds, but the difference is much smaller between the two approaches than in the well-behaved case, for reasons mentioned above.

4.2.3 Discussion

As seen above, results from the metric do not reveal much about the considered experiments. As discussed in Subsection 3.4.1, the presence of several peercasters as well as time shifting prevents having a precise idea of how close the values are to a theoretical optimum. Additionally, the relative abundance of bandwidth means that towards the end of the experiment, when fewer blocks still have to be distributed, efficiency in distributing blocks matters less and less. As a result, the metric values at the end of the experiment do not display significant differences between the two approaches.

The differences observed in the flash crowd scenario, both in the selfish and well-behaved cases, are evidence that the proposed approach can provide additional performance in a highly dynamic scenario, where peers join and leave the system quickly. This is explained by its emphasis on avoiding congestion when uploading and downloading, which allows it to more quickly leverage new peers' bandwidth.

Contrary to what was expected, it does not fare unambiguously better than the existing policy in cases where peers are selfish. This is evidence that the peercaster policy is not able to reduce extreme values of lag in this setting. It is also evidence that the current LiveShift policy, which gives a more reliable stream to peers with a high upload rate, performs relatively well with this type of selfish behavior: as it first fulfills their desires in terms of content, it leaves them free to upload to less attractive peers. On the opposite, by sometimes interrupting the stream of high upload peers, the proposed approach may force them to compensate by trading blocks among themselves, thereby negatively impacting downstream peers more than it helps them.

4.2.4 Applicability to LiveShift

When thinking about the applicability of these results to LiveShift, a number of issues must be kept in mind. Firstly, an implementation would require changes to both the LiveShift application and to the approach itself. Secondly, other incentive-related issues may mitigate the approach's effectiveness.

Potential Implementation Issues

In terms of the information peers exchange, a number of changes to the LiveShift application [9] would be required or beneficial in a hypothetical implementation. For maximal effectiveness, both upload and download bandwidth allocation processes need to have the exact block information for many peers; this would, however, prove too demanding,

especially in terms of the network load it would create, due to the necessity to update that information very frequently. A realistic compromise would be to reduce the segment length and replace the block with the segment as a content unit. This would somewhat reduce the approach's effectiveness compared with using blocks, but more information would be retained than with larger segments; the required frequency of upgrade would also be lower, as segments are not acquired as quickly as blocks. The number of requests received, which measures peers' congestion, would constitute a completely new value to be transmitted. Since there are no discrete steps in LiveShift, a time-limited value would have to be adopted, ideally the number of requests received in the last few seconds.

When choosing who to request from, a downloading peer (requesting peer in [9]) should ideally have request information on all the peers who possess blocks it is interested in. That would represent too many messages, so a hybrid approach could be used instead: the candidates would be ordered according to their level of congestion, and the peer would check block availability and number of requests regularly for those. In addition, it would more or less frequently check these same values on any other peers who possess interesting segments, and replace a candidate if it found a peer with lower congestion than the most congested candidate (depending on how interesting the content of the found peer is). As for the upload behavior, it could be fairly easily implemented by changing the priority rating of peers as already exists in LiveShift, replacing it with the contribution to the payoff function. However, in order to compute that contribution, the uploader (providing peer in [9]) would also need to request the segment information on each of its requesters from the distributed tracker (which handles published segment information in LiveShift). This increase in messages would also have to be taken into account when deciding on the segment length. Finally, peers would have to be implemented so as to change their behavior when they are peercasters: first, they would use a different priority order (the popularity coefficient ω_k), and they would also ideally know of any peer interested in any segment of the stream they originated, then requesting segment information for all of these peers, allowing them to find the least popular peer to which they can grant a slot or request.

All messages currently part of the LiveShift protocol would be sent asynchronously from that updating process, although replies to these messages could contain update information; the peer would simply use its current computed ranking of peers to sort its requesters, candidates, etc., and these messages (e.g. `Subscribe`, `Interested(segment)`, `Granted`, etc.) would always be sent first to the best ranked peer.

Limitations

As seen above, in a LiveShift implementation, the information the proposed approach requires may need to be summarized and/or updated less frequently than the simulation assumes, in order to limit the networking overhead it would cause in terms of additional messaging. It may also be manipulated by peers, e.g. in the number of requests they received, or the exact blocks that they own (as dicussed in Chapter 3). Additionally, in a real scenario, a stable overlay may be of more benefit than in the considered settings, as no scenario with churn was studied. Think in particular of a scenario where the probability of a peer disconnecting is inversely proportional to its longevity in the

overlay; this is expected to be advantageous to a more stability-focused approach such as LiveShift. Lastly, actual scenarios will of course combine more sources of instability than flash crowds, and flash crowds themselves will probably have a slightly smoother profile [1] than those simulated here, with many peers joining and leaving relatively shortly one after the other, but not instantly.

Chapter 5

Summary and Conclusions

5.1 Summary

In this thesis, a motivation for the need for an incentive scheme with self-maximizing peers for the LiveShift system was first presented. Then, a review of the related work was undertaken, and a lack of adequate solutions for those goals was noted.

The first contribution made is an incentive model of a time-shifted video-streaming system, which is represented by a repeated game, and based on [17]. One step of that repeated game is separated in a download and an upload phase. Those two phases are modeled as distinct games taking place sequentially. In the download phase, a congestion game is used, with the costs of requesting from a given peer rising as that peer receives more requests. In the upload game, the value of choosing a given peer to upload to is proportional to its reciprocating upload in the current stage, the value of the content it owns (and potentially can contribute), and to a popularity index which reflects the expected congestion it may face in the future. In this model, an optimal situation is also described, in terms of peer behavior and outcome, albeit in a static and deterministic setting. From the optimal peer behavior, an incentive scheme is defined in terms of rules of behavior for the peercaster, who originates content and therefore is assumed to have as a goal the best performance of the system. Finally, the model encompasses a simple metric of system performance, that depends only on the speed of content distribution.

Secondly, a simulation of that model is implemented, incorporating as well the current LiveShift approach. It allows running experiments varying standard peer behavior (well-behaved or selfish), peercaster policies (LiveShift or the proposed approach), and the amount of change in the overlay (stable or with flash crowds). The results of that simulation show that the metric proposed as a part of the theoretical model does not adequately distinguish between the chosen scenarios. Instead, lag results are presented (as in [9]) which show a clear advantage for the proposed approach in well-behaved scenarios, especially if flash crowds join and leave the system.

5.2 Conclusions

After presenting a model and evaluating it through simulation, conclusions can be presented with regards to the model itself, its effectiveness in the simulated settings, and these results' applicability to LiveShift.

5.2.1 Model

The solving process described in the proposed model does not use what would be the standard game-theoretic approach in a repeated interaction, i.e. a solution of the repeated game. Such a solution would be computationally costly, and impossible in a dynamic system. Therefore, an approach where each step of the system is solved separately is preferred; however, the peers' interest in the outcome of subsequent steps is reflected in their payoff in the present step. This allows the model to still take into account the repeated nature of the interaction taking place in the system.

In terms of peer preferences, the model uses congestion games for its download behavior part. These are well understood and used to study various P2P systems ([21],[17]); their simple cost function that is based purely on the level at which a resource is used is a good fit for a setting where a choice of resources must be made. The upload behavior part, however, does not use a standard game model. As such, its mathematical properties are not known. Its payoff function (Definition 4) takes direct reciprocity and the popularity coefficient ω_k as an index of future congestion into account. Although an evaluation of contributions from the target peer is an expected component, popularity as measured here is not. It does contain information that is relevant to future congestion, including the upload rate; however, it is certainly not the best of such measures. As such, the model presented in this thesis cannot be considered the most effective application of game theory to LiveShift.

Additionally, the model does not deal definitively with issues of information manipulation. For its purposes, it was assumed that peers would truthfully reveal the values required for the computation of the different payoff functions. Although it may be in peers' interests to do so in certain cases, there are still instances where manipulation would be advantageous. This is not dealt with completely in the model.

5.2.2 Simulation results

Under the tested conditions, the proposed approach does in general grant an advantage over current LiveShift policies in terms of system performance. This advantage is however not as strong in all of the tested scenarios, as it is sometimes reversed for a large minority of peers. The performance difference is strongest when the peers are well-behaved, and/or when flash crowds exist in the system, which is common in actual streaming traffic [1]. This would apply e.g. to LiveShift in its default configuration state, where peers are expected to be obedient to the protocol.

With selfish behavior, the difference, when there is one, is mostly positive; for many peers, the approach simply causes no change in performance compared to current LiveShift policies. In the selfish, flash crowd scenario, half of the peers experience a performance increase and most of the other half are not impacted at all. However, the performance difference in the selfish case is not as strong as in the well-behaved case, and does reverse where very high values of lag are concerned. This reduced impact is consistent with the reduced difference between the proposed approach and LiveShift policies in the selfish case, where only peercaster policies differ. It can therefore be concluded that the proposed approach provides an overall performance increase in the selfish case as well, albeit not with the same level of certainty as in the well-behaved case.

5.2.3 Applicability to LiveShift

As discussed in Sections 4.1.1 and 4.2.3, the simulation does not fully reproduce the LiveShift protocol, and does not simulate all aspects of a realistic case. Additionally, an implementation of the approach in LiveShift may have limitations in terms of the speed with which certain information is updated, because of network congestion concerns. As far as settings similar to the ones described:

– there is evidence that the proposed approach would provide a performance boost in a well-behaved setting, both in the stable and the flash crowd scenario;
– in the selfish setting, the evidence allows the presumption that any effect on performance is more likely to be positive. However, because of the limitations outlined above, the existence of that effect is not as certain.

5.3 Open Questions

Several aspects of the matter dealt with in this thesis bear open question or offer avenues for future work. First off, an implementation and evaluation of the proposed model or a similar one in LiveShift would help validate or invalidate the conclusions reached here. This implementation should also be evaluated under additional scenarios not covered in this thesis. The proposed model should be further investigated as well, in particular the upload payoff function and what alternatives to it could be. The question of how to ensure truthful information is revealed by peers (e.g. when it comes to upload rate) in LiveShift also remains open.

Bibliography

[1] Ignacio Bermudez, Marco Mellia, and Michela Meo. Passive Characterization of SopCast Usage in Residential ISPs. In *Proceedings of the 2011 IEEE International Conference on Peer-to-Peer Computing*, IEEE P2P '11, pages 1–9. IEEE, September 2011.

[2] Thomas Bocek. TomP2P - A Distributed Multi Map. `http://www.csg.uzh.ch/publications/software/TomP2P`, 2009.

[3] Thomas Bocek, Wang Kun, Fabio Victora Hecht, David Hausheer, and Burkhard Stiller. PSH: A Private and Shared History-Based Incentive Mechanism. In *Proceedings of the 2nd international conference on Autonomous Infrastructure, Management and Security: Resilient Networks and Services*, AIMS '08, pages 15–27, Berlin, Heidelberg, July 2008. Springer-Verlag.

[4] Yan Chen, Beibei Wang, W.S. Lin, Yongle Wu, and K.J.R. Liu. Evolutionary Games for Cooperative P2P Video Streaming. In *Image Processing (ICIP), 2010 17th IEEE International Conference on*, pages 4453 –4456, September 2010.

[5] Xiaowen Chu, Kaiyong Zhao, Zongpeng Li, and A. Mahanti. Auction-Based On-Demand P2P Min-Cost Media Streaming with Network Coding. *Parallel and Distributed Systems, IEEE Transactions on*, 20(12):1816 –1829, December 2009.

[6] Yang-hua Chu, John Chuang, and Hui Zhang. A Case for Taxation in Peer-to-Peer Streaming Broadcast. In *Proceedings of the ACM SIGCOMM workshop on Practice and theory of incentives in networked systems*, PINS '04, pages 205–212, New York, NY, USA, September 2004. ACM.

[7] B. Cohen. Incentives Build Robustness in BitTorrent. *Workshop on Economics of Peer-to-Peer systems*, May 2003.

[8] Michal Feldman and Noam Nisan. Topics on the Border of Economics and Computation, Lecture 2. October 2009.

[9] F. Hecht, T. Bocek, R. G. Clegg, R. Landa, D. Hausheer, and B. Stiller. LiveShift: Mesh-Pull P2P Live and Time-Shifted Video Streaming. Technical report, Communication Systems Group (CSG), Institute of Informatics (IFI), University of Zurich, Binzmühlestrasse 14, CH-8050 Zürich, Switzerland, September 2010.

[10] Hao Hu, Yang Guo, and Yong Liu. Mesh-Based Peer-to-Peer Layered Video Streaming with Taxation. In *Proceedings of the 20th international workshop on Network and operating systems support for digital audio and video*, NOSSDAV '10, pages 27–32, New York, NY, USA, June 2010. ACM.

[11] Yan Huang, Tom Z.J. Fu, Dah-Ming Chiu, John C.S. Lui, and Cheng Huang. Challenges, Design and Analysis of a Large-Scale P2P-VOD System. *SIGCOMM Comput. Commun. Rev.*, 38:375–388, August 2008.

[12] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The Eigentrust Algorithm for Reputation Management in P2P Networks. In *Proceedings of the 12th international conference on World Wide Web*, WWW '03, pages 640–651, New York, NY, USA, May 2003. ACM.

[13] Idit Keidar, Roie Melamed, and Ariel Orda. EquiCast: Scalable Multicast with Selfish Users. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, PODC '06, pages 63–71, New York, NY, USA, July 2006. ACM.

[14] D.S. Menasché, D.R. Figueiredo, and E. de Souza e Silva. An Evolutionary Game-theoretic Approach to Congestion Control. *Performance Evaluation*, 62(1-4):295 – 312, 2005. Performance 2005.

[15] Roger Myerson. *Game Theory*. Harvard University Press, Cambridge, 1997.

[16] E. Mykoniati, R. Landa, S. Spirou, R. Clegg, L. Latif, D. Griffin, and M. Rio. Scalable Peer-to-Peer Streaming for Live Entertainment Content. *Communications Magazine, IEEE*, 46(12):40 –46, December 2008.

[17] Giovanni Rossi, Gabriele D'Angelo, and Stefano Ferretti. Multistage Congestion Games for Live Streaming. In *Computers and Communications (ISCC), 2010 IEEE Symposium on*, pages 830 –833, June 2010.

[18] Giovanni Rossi, Stefano Ferretti, and Gabriele D'Angelo. Equilibrium Selection via Strategy Restriction in Multi-Stage Congestion Games for Real-time Streaming. Technical Report 12, April 2009.

[19] Ralf Steinmetz and Klaus Wehrle. *Peer-to-peer systems and applications*. Springer, 2005.

[20] Xiao Su and Suchreet K. Dhaliwal. Incentive Mechanisms in P2P Media Streaming Systems. *IEEE Internet Computing*, 14:74–81, October 2010.

[21] Subhash Suri, Csaba D. Toth, and Yunhong Zhou. Selfish Load Balancing and Atomic Congestion Games. *Algorithmica*, 47:79–96, January 2007. 10.1007/s00453-006-1211-4.

[22] Éva Tardos and Tom Wexler. Potential Games and Congestion Games. In *Algorithmic Game Theory*, pages 497–498. Cambridge University Press, 2007.

[23] Jean Tirole and Drew Fudenberg. Extensive-Form Games. In *Game Theory*, pages 67–106. MIT Press, 1993.

[24] Jean Tirole and Drew Fudenberg. Nash Equilibrium. In *Game Theory*, pages 11–28. MIT Press, 1993.

[25] Jean Tirole and Drew Fudenberg. Strategic-Form Games. In *Game Theory*, page 4. MIT Press, 1993.

Abbreviations

P2P	Peer-to-Peer
GT	Game Theory
PoA	Price of Anarchy
PoS	Price of Stability
NE	Nash Equilibrium
BT	BitTorrent

Glossary

Cost function (peer, download) In the congestion game modeling the download behavior, the cost incurred by requesting from a peer is proportional to the number of queries that peer already received. It reflects the lower probability of a successful request to a highly congested peer.

Payoff function (peer, upload) In the upload behavior game, the payoff gained by uploading to a peer is proportional to how much that peer uploads in return in the same step of the game, the interesting content that peer has, and its popularity.

Player (also Selfish Peer) A non-peercasting peer in LiveShift, who receives and consumes stream data. Will be selfish or well-behaved depending on scenarios.

Popularity Coefficient, computed on the basis of a peer's upload rate and owned video blocks (depending on other peers' interest for these blocks) that reflects a peer's relative expected congestion, i.e. its ability to satisfy the requests it is likely to receive.

Utility (single block) Value attributed by a peer to a block, as a function of that peer's interest for that block. Helps quantify global interest for that block (see Popularity).

List of Figures

List of Tables

Appendix A

Installation / Usage Guidelines

A.1 Simulation

1. Copy the `simulation` folder on the CD to a new folder;
2. Open that new folder as a workspace in eclipse – version used: 3.6.2 (Helios SR2), JRE 1.6;
3. The `simulation` project contains the main class `SimulationMain` in package `controller.runs`;
4. Run that class to run the experiments. In the code, the following can be set:
   - the `repeats` variable at the beginning of the `main` method determines the number of times each experiment must be run;
   - the static field `maxTh` to limit the number of threads used for running the experiments.

A.2 Graph generation

Once the simulation has been run, graphs can be generated from the data. This is done using the file `examples+macros.xlsm` (see Appendix B):

1. Make sure both `examples+macros.xlsm` and one of the `final.csv` files are open in Excel;
2. in `final.csv`, run the `Graphs` macro (View tab -> macros in Excel 2007 / 2010);
3. some formatting (font size, color and style of data series) must be applied manually on the graphs.

Appendix B

Contents of the CD

- Thesis LaTeX source code: `Thesis` folder.
- Simulation Java source code (as eclipse project): `simulation` folder.
- Raw result data: `results.zip` file.
- Graphs from the data: `graphs` folder. Contains the `examples+macros.xlsm` file.
- Intermediate presentation in the eponymous folder.
- Articles used in the related work and otherwise cited. Names correspond to the keywords defined in the `bibliography.bib` file in the `Thesis` folder.