Incentive Mechanisms for Live Media Streaming in P2P Networks

Thesis report by

Aikaterini ANDREADOU

Encadrants
Olivier FOURMAUX
Bénédicte LE GRAND

Université Paris 6, Pierre et Marie Curie
Master 2, Computer Networks

September 2007

Contents

1. Introduction

Since its creation and especially nowadays, Internet has excelled its purely technological aspects and it has become a complicated social phenomenon, a virtual community where users interact and all kinds of behaviours can easily be met. The notions of anonymity, invisibility and absence of authority are closely linked to the essence of Internet, which is considered by some as a modern virtual democracy. The P2P networks and the content sharing applications based on them constitute an important (if not the most) trend in today's internet traffic and in its users' habits. In fact, the common use of the Internet today is so content oriented that there have been thoughts and studies on adapting the routing itself to a more content based pattern. Starting by Napster, an avant-garde content sharing application, the evolution of P2P has led to today's mainstream extremely popular live streaming programs, like PPLive, CoolStreaming etc. The distributed nature of these applications is a technological choice aiming at satisfying the growing users' demands, which could not any more be met by the capacity of one or multiple servers. So, in the absence of an IP multicast service supported by the network infrastructure (routers), an alternate solution had to be found deployed at the application layer. In this new approach, the nodes are organized into an overlay topology, they form a new virtual network over the internet, whose logical links may correspond to many actual physical links of the underlying network. This distributed nature has also a political aspect: it constitutes a counter measure or a defence against authorities who would menace the existence of these communities. Since there is no unique server or point of entrance, there is no simple way of shutting them down. In conclusion, we can say that it is at this distributed notion that lies the revolution of P2P but also its complexity, with which we have to deal every time we try to introduce a new feature.

What's more, the complicated user interactions in these virtual environments have aroused the curiosity of psychologists and sociologists, who have tried to explain how the sentiment of importance inside a community can work as a stimulus to unselfish contribution for a lot of users. Naturally after that, programmers and engineers thought they could use these interesting aspects of human behaviour, extend them to online societies and create incentive mechanisms for contribution, co-operation and altruism among fellow peers. Latest researches have also proposed the formation of local communities based on geographic criteria thinking that it 's easier for users to trust each other if they have been or have the chance to be physically acquainted. So, again the level of unity of P2P communities is a matter of trust. It's either that or the enforcement of strict and limiting rules aiming at minimising the vulnerabilities of the system, which tend to be exploited by the free-riders, users whose aim is to download the maximum amount of data without contributing at all to the system.

1.1 Goal

The goal of this work is to examine how different incentive techniques affect the performance of live streaming in a P2P network. To reach this goal, we had to study the state of the art content sharing and live streaming applications in order to understand their functionality and also the different "families" of incentive mechanisms, so as to be able to implement them, compare them and draw rational conclusions. A summary of this extensive study, containing only the crucial elements necessary to come to a profound understanding of the work done, can be found in the first section of this document, along with comments and a comparison of some of the more popular applications. The conclusion is twofold: In terms of overlay architecture, it has become obvious that single tree structures suffer severely from

node failures, since they don't share the load equally among the peers. On the other hand mesh topologies react better to network variations but they become difficult to manage for large numbers of users and there is a great deal of messages exchanged only to keep track of one's neighbours.[20] Finally, multiple trees are very flexible but their function is also very complicated, especially when it comes to maintaining large numbers of trees for each peer. So, a solution that seems appropriate is the simplicity of a single tree structure combined with the flexibility of local meshes formed along the tree. Now, in terms of incentive mechanisms, live streaming is a very demanding application so special handling is necessary. Live streaming applications are very sensitive to delays in two ways: Firstly, “liveness” is a crucial property of the videos distributed, which means that when a user logs in to watch a football match, for example, he will not bear a considerable delay before the actual beginning of the streaming. What’s more he will not bear pauses, interruptions or disconnections. Secondly, once the streaming begins, the video quality is highly sensitive to packet delays, all pieces must arrive before their play out time, if not they are useless and they will be rejected. So, a considerable number of lost or late packets will severely damage the perceived video quality. It’s for these reasons that a live streaming application demands that everybody contributes, it cannot work on the basis of a minority of regular uploaders, like file sharing systems, and cannot sustain large numbers of free riders. That’s why the existing solutions don’t seem to be enough, a simple tit-for-tat policy doesn’t manage to sustain regular contributions by all the peers, a micro payment mechanism is usually very restrictive and not very welcomed by end users and a reliable reputation system usually demands the existence of a central authority. So, the proposed solution is oriented towards a differentiated services system based on a reputation scoring function, which as a whole has proved to be very suitable for the live streaming paradigm. It's upon this analysis in total that the actual choices of the implementation and the proposed mechanism are based.

1.2 Difficulties and Choices

At the beginning of this work, the first question to be answered was in what way and with which means the incentive mechanisms will be studied? In the absence of an actual experimental P2P network on which these algorithms could be deployed, the only solution was the simulation. And so, at this point the major dilemma arises: should we use an existing simulator or build a new one from scratch? In order to answer appropriately, we examined some of the existing simulators, like the SimPy in Python, the PeerSim in Java and the PeerfactSim created by the University of Darmstadt, and we realised that all these event driven simulators were more concentrated on the simulation of P2P protocols with many interesting characteristics as synchronization handling, packet formats and message exchange. In reality, these simulators were rather complicated in a way not useful to the goal of our work, as the aim was to study the effects of incentive algorithms without very much messing with the P2P network infrastructure, and on the same time they didn't supply much support and modules to simulate the live streaming aspect, which was necessary. So, we decided to build a new application in Java in order to simulate the algorithms concentrating on the details we considered more important. The relative success or failure of this choice has yet to be proved by the results that will be presented.

1.3 Organization of the work

In this work, we present an overview of the problem of incentive mechanisms in live streaming. The remainder of this report is organized as follows.

First, we examine the most important existing applications in this domain starting with BitTorrent protocol, which, though not appropriate for live streaming, has set the fundamentals for all its successors. We continue with DONet, Bullet and Pulse, which offer us an in depth insight in the function of this kind of applications with its temporal representation. Then we do a comparison of these three applications, in order to draw conclusions concerning their advantages, disadvantages and similarities and also explore their functionality so as to clarify the way the implementation choices are reflected on the final architecture of the produced overlay network. Secondly, we do a quick summary of the major challenges posed when deploying incentive techniques in a P2P system and also of some of the solutions and remedies to these problems. Then we present the three major incentive families, systems based on micro-payment, reputation and differentiated services. The distinction among these techniques is not always clear and obvious, so we try to understand the virtues and vulnerabilities of every technique concerning different cases, and finally we show that a differentiated services system based on a reputation function seems to be the most appropriate solution for the live streaming problem.

The second section of this document concerns the simulations and the results in form of figures and tables. First, we present the configuration, the parameters and the choices made concerning the implementation of the simulator, we describe the characteristics of the nodes, the way they are connected and their knowledge of their environment, the parameters we chose to vary and we also added a pseudo code of the actual core of the algorithm. Then, the actual results of the simulations follow. To begin with, we test two simple scenarios, in the first one each node chooses as his next uploader the peer with the bigger upload capacity, in the second one he chooses the one with the least bitmap resemblance. Afterwards, we launch a simulation of the BitTorrent choke algorithm to find out that its performance is exceptional, though its resistance to free riders is rather low. Then we examine the performance of the mechanism we proposed, inspired by the study of the state of the art algorithms done in the first part, which presented similar results to BitTorrent. Finally, we examine a scenario with one rich source and a case where a considerable bufferisation delay was introduced. Extensive comments, comparison and conclusions derived from these simulations can be found in this second section.

2. State of the Art - Background

2.1 State of the Art P2P Content Distribution Systems

2.1.1 BitTorrent

Although the main subject of this work is live streaming, I think that it is essential to present the main functions of BitTorrent, the most successful file sharing predecessor of today's live streaming applications. Contrary to Gnutella or KaZaa, who aim mainly at the fast localisation of fellow nodes in possession of a certain file, BitTorrent's primary goal is the fast replication of a large file by a group of nodes. It was essentially the first application to combine the notions, ideas and policies, which would allow it to be more wide spread than any file sharing application until then. These innovative new elements are presented in the following tables. Most importantly, it introduced the idea of the tracker, which helped to attain a higher level of decentralisation and distribution of resources than before, although it still remains a central entity, prone to cause problems to the total structure, when confronted with large numbers of simultaneous users or potential attackers.

torrent	The set of peers cooperating to download the same content
tracker	The only centralized component of the system. It keeps track of all the active peers and stores relative information.
seed	A peer that has already downloaded all the pieces of a file and is sharing it with others.
leecher	A peer that hasn't already downloaded all the pieces of a certain file.
chunks **blocks**	A large file is divided in smaller pieces, called chunks (256 kbytes), in order to facilitate replication by multiple users. The chunks are also divided in smaller pieces, called blocks (15 kbytes), which constitute the data transmission unit in the system.

Table 1 (up) : Basic BitTorrent Elements

Table 2 (down) : Basic BitTorrent Policies

Peer selection strategy: **Choking or tit-for-tat** **algorithm**	This strategy is supposed to encourage cooperation and discourage the free riders and it is used to determine which peers to exchange data with. The general idea is that once every rechoke period (typically set to 10 seconds), a peer selects a certain number of its fastest downloaders and uploaders and reciprocates only with them choking the rest.
Optimistic unchoke	This procedure is an exception to the latter case and it takes place every 30 seconds. It means that a peer is unchoked at random without considering his capacity in order to discover new peers, who could offer better service, and to help new comers to obtain some pieces so as to function well in the system.
Chunk selection strategy: **Rarest first**	The goal of this policy is to maximize the entropy of each chunk in the system and in consequence to make sure that each peer can always find a missing piece owned by another peer. The rarest first policy consists of always trying to serve the rarest chunk among those demanded by the nodes in the peer set.
Random first policy	An exception to the latter policy is the case where a node has just logged in, so he must urgently download a few chunks to be able to be well integrated in the system. In this case he selects to download a chunk at random.

The interactions among the peers in a BitTorrent system are determined by a group of policies and rules, which are essentially the core of the BitTorrent protocol. They are summed up in table 2.

Legout, Liogkas, Kohler and Zhand in [1] have made a very interesting analysis showing the effects of the different BitTorrent mechanisms on the motivations of sharing and clustering. They have deduced that if there is an initial seed, capable of supporting a high upload rate, the peers of the same upload capacity tend to form clusters and reciprocate with peers of the same cluster. This phenomenon becomes evident in the following figure: The darker squares represent longer unchoke periods. The peers from 1 to 13 have an upload limit of 20 kbps, the peers from 14 to 27 50 kbps and those from 28 to 40 200 kbps, like the initial seed. It becomes evident that the three peer categories form clusters and interact within them, except the case of an optimistic unchoke. We can also see that often slow nodes unchoke medium nodes. However, this behaviour isn't mutual, medium pairs rarely unchoke slow ones because they cannot benefit from them so there is no interest in doing so.

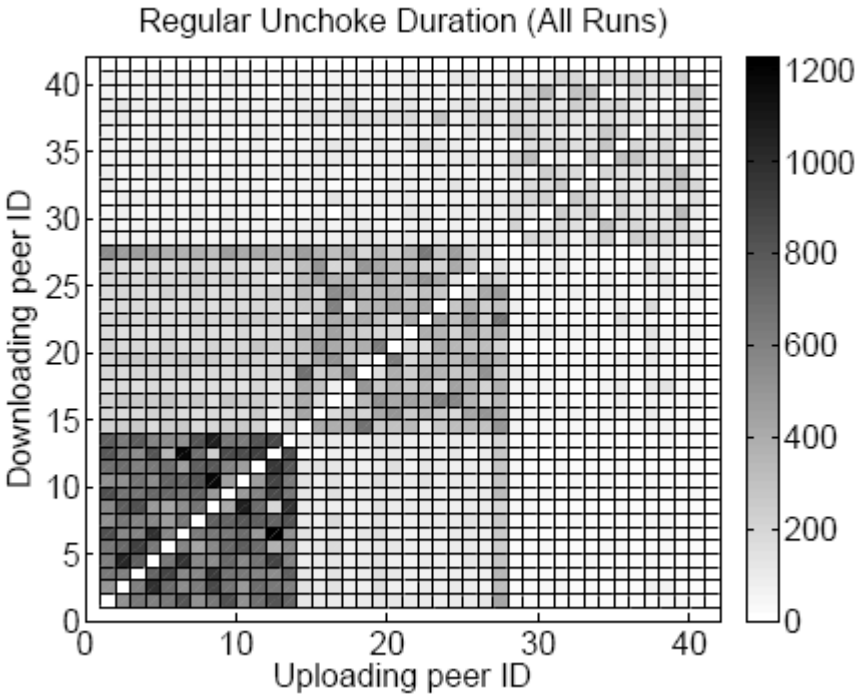


Figure 1: "Clustering and Sharing Incentives in BitTorrent Systems", Legout, Liogkas, Kohler, figure 1

An interesting approach is that of Vlavianos, Iliofotou and Faloutsos in [2] who try to determine the minimum necessary changes in order to transform BitTorrent into a protocol capable of streaming. The authors specify that the proposed solution concerns the Video On Demand ant not the Live Streaming, because in the latter case the packets are created dynamically, they are not already ready at the beginning of a session, so the problem becomes too complicated to confront with a mere modification of the existing protocol. The main idea of their proposition is to modify the chunk selection algorithm attributing a high priority to pieces who are going to be soon reproduced by the media player: In essence they introduce two categories of missing pieces, the High Priority Set, which includes the necessary pieces for the generation of a small duration of the video that follows, and the Remaining Pieces Set, which includes all the pieces that haven't been downloaded yet and don't belong to the High Priority Set. This way, each client decides to download a piece from the High Priority Set with a probability p (p=0.8 by default)

and a chunk from the Remaining Pieces Set with a probability 1-p. This new BitTorrent based application is called BitoS.

Since most of the users started to have access to high bandwidth connections, many live streaming applications have appeared. I am going to present some of them.

2.1.2 DONet

DONet is a Data-driven Overlay Network with no specific structure, where the availability of data is the one that determines the exchanges and the flow directions. The availability of each segment of the video is represented by a Buffer Map, periodically exchanged among the nodes. [3]

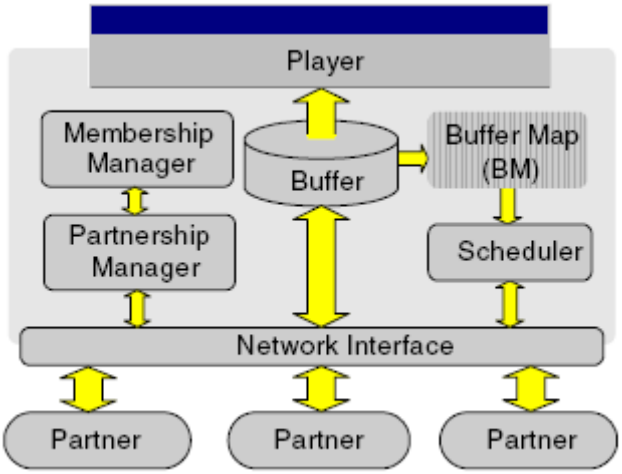


Figure 2: "CoolStreaming/DONet: A Data-driven Overlay Network for Peer-to-Peer Live Streaming", Xinyan Zhang, Jiangchuan Liu, Bo Li, Tak-Shing Peter Yum, figure 1

Three key modules

- Membership Manager : Maintains a partial view of other overlay nodes
- Partnership Manager : Establishes and maintains partnerships with other nodes. A partner is a node with which we are engaged in regular exchanges of data.
- Scheduler : Schedules the transmissions of video data

Joining Algorithm

- The new node contacts the origin node.
- The origin node randomly selects a deputy node from its Cache and redirects there the new node.
- The deputy supplies a list of partner candidates.
- The new node contacts the candidates to establish its partners in the overlay. (optimal number of partners=4)
- While connected, the new node periodically generates a membership message to announce its existence and help the origin node update its Cache. Gossip protocol used.

Scheduling Algorithm

- Calculate the potential suppliers for each segment.
- Determine the supplier of each segment starting from those with only one potential supplier.
- Among them, chose the one with the highest bandwidth and enough available time.
- Problem: Free-riding risk -> A node can advertise conservative buffer maps to avoid contribution. Risk: Can he do that without harming his own performance?

Node Departure and Failure Recovery

- Graceful departure: The departing node issues a departure message.
- Node Failure: A partner who detects the failure issues the message on his behalf.
- The departure message is gossiped to the rest of the network.[4]

Partnership Refinement

Each node periodically establishes new partnerships in order to maintain a constant number of partners and to explore the network for better service. The potential partners are chosen based on a score which measures their contribution to the node per unit of time. Intuitively, nodes with high outbound bandwidth and a lot of available segments will be preferred.

Overlay Refinement

Because of the use of a gossip protocol for the exchange of messages among the nodes, the architecture of the underlying physical network is not taken into account for the formation of the overlay. As a result, there is a mismatch between the P2P overlay and the physical structure of the network. The authors of [5] propose the use of a Triangulated Heuristic in order to predict the distance between any two peers and subsequently choose the adjacent peers as partners. This strategy manages in fact to improve the performance of DONet with the disadvantage of a little additional control message overhead.

2.1.3 BULLET

Bullet's innovation lies in the fact that it layers a high-bandwidth mesh on top of an arbitrary overlay tree. The mesh is formed by perpendicular links across the overlay, which naturally augment the available bandwidth. Hence, each node receives a parent stream from its natural parent in the tree and a number of complementary streams from chosen peers in the overlay.

For example, in the following figure, A has sufficient bandwidth to deliver only 3 objects per time unit to his child D. However, D locates nodes C and E, who are able to transmit missing objects, increasing its inbound bandwidth from 3 to 6 objects per time unit.

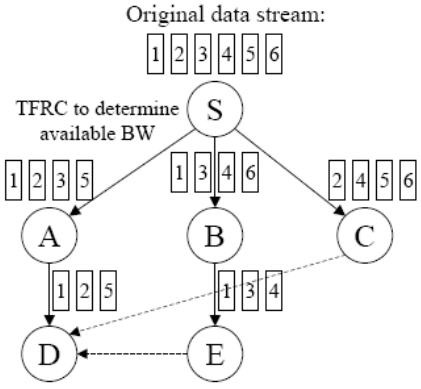


Figure 3: "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh", Kostic, Rodriguez, Albrecht, Vahdat, figure 1 "**High-level view of Bullet's operation**"

RanSub

- RunSub is used to distribute uniform random subsets of global state to all nodes using Collect and Distribute messages. The goal of this procedure is to locate remote nodes with interesting content and good bandwidth.
- Collect messages start at the leaves and propagate up to the tree, leaving state at each node along the path. They contain a random subset of the descendants of each node along with an estimate of its total number of descendants.
- Distribute messages start at the root and travel down the tree, using the information left at the nodes during the collect phase to distribute uniformly

random subsets of remote nodes to all participants.

- Finally, the distribute set contains a random subset representing all nodes in the tree except for those rooted at the particular child. This means that a Bullet node attempts to recover missing data from any non-descendant node, not just ancestors, thereby increasing system scalability.[6]

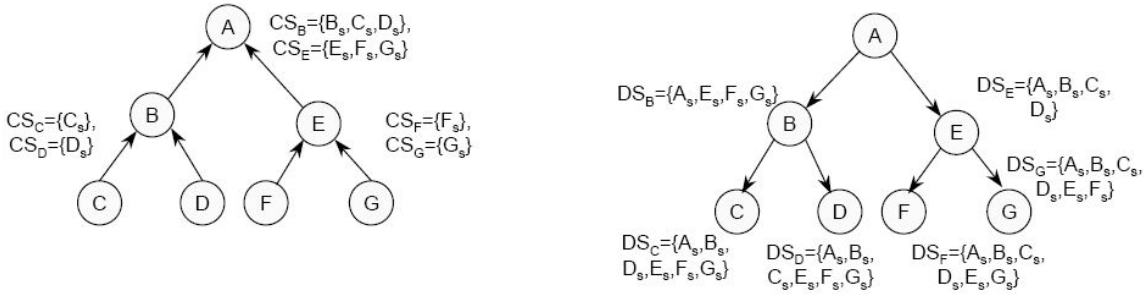


Figure 4: ": High Bandwidth Data Dissemination Using an Overlay Mesh", Kostic, Rodriguez, Albrecht, Vahdat, figure2, "**The two phases of the RanSun protocol**"

Finding overlay peers

Summary tickets are used to represent the working set of each node. Working sets contain the sequence numbers of the packets received by the node over a period of time. This way, upon receiving a random subset of remote nodes at each distribute phase, each Bullet node will choose to peer with the node having the lowest similarity ratio to its own summary ticket. Afterwards, it sends to the remote node a peering request containing its Bloom filter.

Recovering data

Assuming it has available bandwidth for the newcomer, a recipient of the peering request installs the received Bloom filter and will periodically transmit keys not present in it to the requesting node. The requesting node will refresh its installed Bloom filters at each one of its sending peers periodically.

Making data disjoint

Limiting factors are used as feedback from children to determine the best data to stop sending when a child cannot handle the stream, in order to assure that it is with the same probability that each node owns a particular piece. That means that a Bullet parent sends different data to its children so that each data item will be readily available to nodes spread throughout its subtree.

Improving the Bullet Mesh

A node can adapt dynamically the number of its active senders and receivers to improve its performance. Each node periodically drops and replaces one or more peers who are delivering the least amount of useful data to it. In this way, it tries to keep the best senders. Likewise, each node periodically evaluates its receivers and drops the ones acquiring the least portion of its bandwidth, in order to keep the best receivers. [7]

2.1.4 PULSE

One of the virtues of PULSE is its temporal representation (figure 5). This way, a reference system is created. The variables that grow at a fixed rate over time, like Td, are associated with stationary points, while all the others change their position according to their relative instantaneous speed expressed in terms of time or packet rate.[8]

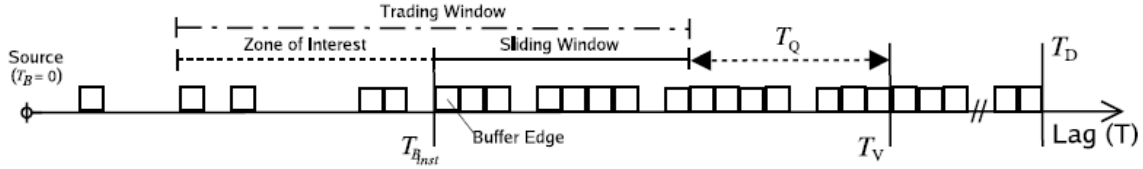


Figure 5: "PULSE, a Flexible P2P Live Streaming System", Fabio Pianese, Joaquin Keller, Ernst W. Biersack, fig1, "**A PULSE node's data buffer**"

Parameter	Description
Tb_{ist}	Average lag of the chunks the peer is requiring. It is placed in the middle of the trading window and it quickly fluctuates.
Tb_{avg}	Average value for a series of $Tb_{ist.}$
Tk	Lag difference between the chunk at the end of the trading window and the chunk at the end of the peer's buffer.
Td	Lag of the chunk at the end of the peer's buffer.
Tq	Lag of the chunk at the end of the trading window.
Trading Window	It includes the chunks the peer is trying to obtain from the other peers. Its size is double than that of the sliding window.
Sliding Window	It is the oldest part of the trading window. It contains the chunks, having a lag greater than the chunk at $Tb_{ist,}$ the peer is trying to retrieve. It will left shift, allowing the trading window to slide, when it contains a sufficient number of chunks.
Zone of Interest	It is the newest part of the trading window. It contains the chunks, having a lag smaller than the chunk at $Tb_{ist,}$ the peer is trying to retrieve

Table 3: "The Pulse System: A new P2P prototype for live streaming", Thesis de Diego Perino, table 3.1, "**Buffer's parameters**"

In PULSE, there are three levels of knowledge about other peers in the system, which are illustrated in table 4, and each one of them has a distinct goal.

- Blue Set: keeping at hand a small and up-to-date list of nodes who share the same streams we are interested in
- Red Set: these peers are the preferred target of a node's attempts to get its missing pieces and with whom it aims at establishing a Missing data exchange
- White Set: keeping track of all the peers we have met, up to a maximum size

	IP address	TCP port	UDP port	Tbavg	Td	chunk buffer edge	Trading window bitmap	RTT
White	+	+	+					
Blue	+	+	+	+	+			
Red	+	+	+	+	+	+	+	+

Table 4: Levels of knowledge

	Overlapping Trading Window?	Tb	goal	consequence to the structure of the network
Missing set	yes	near	trade data to fill any hole around Tb and keep the window sliding	local meshes are formed
Forward set	no	far	allow distant nodes to approach the source	mobility from the back to the forth of the network

Table 5: Two groups of peers for data exchange

There are also two groups of peers for data exchange and four behavioral modes. The details are summed up in tables 5 and 6.

	Target recovery mode	Target group of peers	Goal – active behavior	Fairness Mechanism	Scores	Passive behavior
Friend	Missing	Missing	Main data exchange	Immediate tit-for-tat	Fi	Respond always
Normal	Forward/ Missing	Red	Retrieve data, adjust Hi	Cumulative tit-for-tat	Hi	Respond depending on Hi
Sloppy	Forward	All	Remind peers of their debt	Cumulative tit-for-tat	Hi	Respond depending on Hi
Fast	Forward/ Missing	Red/Blue	When in dire need of data	Cumulative tit-for-tat	Hi	Respond depending on Hi

Table 6: Bahavioral Modes

- Immediate tit-for-tat: BitTorrent-like
- Cumulative tit-for-tat: You don't have to reciprocate immediately. You remain indebted and receive worse quality until you do.
- $$\mathrm{H}\iota = \frac{Cumulative\_Amount\_of\_Data\_\mathrm{Re}ceived\_from\_i}{Cumulative\_Amount\_of\_Data\_Sent\_to\_i} \Big| except\_FRIEND\_exchanges$$
- $$Fi = 1 + \frac{Average\_Inco\min g\_Bandwidth\_From\_i}{SBR} \Big| only\_for\_FRIEND\_exchanges$$
- Conditions to choose a node as a Friend: 1.He must have high Hi 2.He must belong to the Red Set 3.His Tb must be in the range of our Interest window 4.(optional) His inbound and outbound bandwidth must be greater than a certain threshold.[12]

2.1.5 Comparison

We can generally say that all modern P2P applications for live streaming have a hybrid nature. On one hand, a simple tree structure has proved to be inefficient, since it reacts very badly to failures, it doesn't take advantage of all the available bandwidth and it doesn't scale. On the other hand, it is impossible to form a really equally distributed mesh since the source of the data is always the only node that diffuses the stream, so its role is different. So, among the three examined applications, we can say that Pulse is more tree-like, DONet is more mesh-like and Bullet is somewhere in the middle. The similarities and differences of these three applications are summed up in the tables that follow.

	Provider of the stream	***How the local knowledge is obtained***	***Parallel Download***
Pulse	Nodes, members of the red set, who are chosen as friends and moved to the missing set	Three levels of knowledge, white, blue, red, acquired through polling messages	high

	Provider of the stream	***How the local knowledge is obtained***	***Parallel Download***
Donet	Nodes handled by the membership manager who are chosen as partners by the partnership manager	Gossip protocol used to periodically send membership msgs and exchange the buffer map	high
Bullet	Mainly the parent of the node and some perpendicular nodes	RanSub used to distribute subsets of global state using Collect and Distribute messages	medium, the parent is the main source

Table 7: Main Characteristics

	Resistance to failures	***Topological locality***	***Bandwidth optimization***	***Mobility inside the overlay***
Pulse	High, if a node fails, it's easy to replace it with another one from the red set	yes, RTT considered to select a peer	Rich nodes can use their available resources to contribute to the Forward peers	Medium. Friend nodes are stable. Mobility obtained through Forward data recovery
Donet	High, if a node fails, it's easy to replace it with another one proposed by the membership manager	not in the original, yes in the modified version	Nodes with high bandwidth are the first to be chosen as providers for each chunk	High. Providers change as the demanded chunks and their availability change
Bullet	Medium, the parent is the main provider so if he fails, another parent must be found	no	Every node attempts to recover missing data from any non-descendant node in order to fill its inbound bandwidth	Medium. Parent is stable. Perpendicular peers periodically replaced to optimize exchange

Table 8: Adaptability to network conditions

	Free-rider strategy	***Source single point of failure?***	***Fairness mechanism***
Pulse	Advertise conservative trading window, minimize forward slots	No	Immediate tit-for-tat for friends. Cumulative for others
Donet	Advertise conservative buffer map	No, if a dht to a group of source nodes is used.	None
Bullet	Advertise conservative summary ticket	Yes, since it's the unique point of entrance and it's responsible for the Distribute messages	None

Table 9: Fairness Issues

	Feedback to the source	***History score***	***Scheduling algorithm*** ***How to choose partners***
Pulse	None	Fi for friends Hi for others	1. Overlapping trading window 2. high Hi 3.member of the red set

	Feedback to the source	***History score***	***Scheduling algorithm*** ***How to choose partners***
Donet	None	None	Choose the best partner for each missing segment, beginning with the segment with the least suppliers
Bullet	Yes, limiting factors	None	Choose a peer with the lowest similarity ratio to its own summary ticket

Table 10: Fairness Issues II

After this presentation, we can conclude on some main directions to follow for future P2P applications.

- A simple tit-for-tat policy should be extremely efficient in most of the cases. It could also be adopted by DONet and Bullet.
- No complementary roles and responsibilities should be given to the source, apart from its native role as the unique source of the stream, in order to avoid creating a single point of failure of the system. The use of a central component as a tracker should also be avoided as it doesn't handle well flash crowds.
- Some kind of local knowledge of the network and recent memory of the transactions should be kept, like a history score.
- The main goal of the incentive mechanism should be to dissuade users to advertise less data than they actually have to avoid contribution.
- The existence of some kind of clusters or of a central core of credible nodes is inevitable and maybe desirable. The fairness mechanism should not be too strict on these nodes because they keep the system running.

2.2 State of the Art Incentive Mechanisms

In most of the contemporary P2P systems, the users are naturally discouraged to cooperate because contribution directly affects their resources and their performance. In consequence, if a user tries to maximize his performance, his action will have an immediate negative effect on the global performance of the system. So it is necessary to compromise the personal and the collective well being and supply the users with important motivations for cooperation, or else they will naturally become selfish. This compromise is very difficult to make because of the existing challenges in a P2P system:

- Very large and dynamic peer populations, which means that the life duration of a peer in the system can be very short.
- Asymmetric transactions: User A is interested in a content owned by B and B is interested in a content owned by C. As a consequence a user may demand service without being able to reciprocate immediately. So a simplistic tit-for-tat policy isn't appropriate for such a system.
- The majority of P2P systems allow users to constantly change identity (zero cost identity). This characteristic is desirable for the new comers and helps the network grow fast but in the same time it does not penalize malicious users who take advantage of this vulnerability to exploit the system. This behaviour is called whitewashing and constitutes one of the major problems of such applications.
- Most of the P2P systems are unstructured and decentralized. This absence of a central authority imposes the use of complicated solutions in order to manage the system.[13]

However, there are certain techniques that can help us confront these difficulties:

- Combine random and informed peer selection. Informed peer selection will choose already known nodes and will encourage the maintenance of long-term partnerships among them. Random peer selection aims at exploring the network for better service and facilitates the integration of new comers.
- Maintain a short-term distributed history for fellow nodes. Distributed because the asymmetric and time-varying nature of P2P networks would prove a private history to be inefficient. And short-term in order to prevent malicious users from exploiting the system after a period of good behaviour, which would guarantee them a good reputation.
- Adopt a Stranger Adaptive Policy as a defence to white-washing, that means having a flexible policy to strangers that adapts to their recent behaviour. The problem with this solution is that it also penalizes the possible innocent new comers if the majority of the recent strangers have proved to be malicious or selfish.

There are mainly three categories of incentive mechanisms, although the contemporary P2P applications combine them in many ways to derive enhanced performance, so the distinction among them is not always obvious.

Differentiated services: In this kind of mechanisms, the main motivation for the users is the ability to choose reliable and credible peers, who can guarantee them a predictable quality of service. So, every user is free to choose his level of contribution in order to maximize the quality of the perceived video. For example, such a mechanism could allow the peers to select as uploaders nodes with a score inferior or equal to theirs only. In this way a node with a zero score will have a best effort quality of service. If he desires a streaming better than best effort, he must obtain a positive score by contributing to the system.
The score of each user is determined by a scoring function, which could consider and measure the contribution of each user, or the contribution minus his consummation, or even introduce an ageing factor to encourage regular contributions. It could also remunerate more users who upload rare files or strongly demanded ones. The possibilities are endless. [14]

Micro-payment: These mechanisms aim at establishing equilibrium between total downloads and total uploads of each user. A very simple solution is to debit the users for every byte they download and to remunerate them for every byte they upload. The advantage of this method is that it gives users the opportunity to control their debt to the system and even make money if they want. However, these techniques are not generally welcomed by them, especially when there is real money involved, because they are constantly asked to decide if a certain content is worth paying. [15]

Reputation: The simplest use of the reputation mechanisms is to help the peers with a good reputation find each other and cooperate. The main idea is to construct a reputation system to attribute an objective score to each user according to his behavior and afterwards determine his quality of service by this score. It is obvious that here a central entity is necessary to maintain and update the scores, which is not desirable for modern P2P applications. An alternative to the solution of a central entity is that of each peer calculating his own score, afterwards his score being mapped into a percentage by comparing it to the scores of his fellow peers in his little neighborhood and in the end he will be able to know more or less his position in the global distribution and act accordingly.
In the following table, I try to sum up the advantages and disadvantages of each kind of incentive mechanism concerning different P2P application scenarios.

(Next page) Table 11: Comparison

P2P/ Incentive Mechanism	Micro-Payment	Reputation	Differentiated service
Advantages/ Disadvantages	+Debt control +Opportunity to make profit -Problem for the slow or asymmetric connections -Not very popular if there is real money involved because one must constantly decide if a certain content is worth it -Danger of exploitation by users who form coalitions -Difficult to evaluate the rarity or the demand of a file, in order to remunerate more the users serving it	+Flexibility of decision +Better accepted by the users +Defence against malicious users -The absence of a central authority causes problems -A central entity is expensive -It is necessary to maintain a history and exchange polling messages. As a result the cost and the overhead rise.	+The quality of service is directly related to the level of contribution. So there is a motivation to maximise the latter. +No polling messages, so no overhead added -Injustice : Same treatment for the free riders and the new comers -Necessity of a central entity to calculate users' scores. Solution: every user calculates his score locally and does an approximation. -But, integrity problem: the users can lie in order to exploit the system
Files exchange	A variation of this technique is the tit-for-tat adopted by BitTorrent. Mechanism appropriate for these systems because of its simplicity and its facility of implementation. What's more, it allows the slow nodes and the free riders to take advantage of the spare capacity, so that it is not lost. However necessity of a central element ex. tracker	A variation of this technique was adopted by KaZaa where the users were divided in three classes of contribution. Necessity of a central element, or the problem becomes really complicated. This technique isn't as efficient and as appropriate as the tit-for-tat for these systems.	We have found no application of this mechanism in file exchange systems. Which is normal since it introduces a high level of complexity that affects badly the system performance. This complexity is redundant because the system can function fine with simpler algorithms.
Live streaming	This approach or the tit-for-tat isn't sufficient for live streaming. We must use a technique that discourages the free riders and encourages everybody to contribute constantly and continuously, or else the quality of service deteriorates. The vulnerability of this type of mechanisms is that the user is able to accumulate credit and then not contribute at all during a very popular emission when the cooperation is vital.	The reputation mechanisms constitute a very interesting solution to the right direction. However the efficiency of the proposed solution depends on the answers we are going to give to certain questions: Will there be a central entity or not? Will there be a history? Which are the specific parameters maintained by each user? How will reputation affect the decisions to be made? How will reputation determine the quality of service?	It seems that a solution based on the differentiation of the quality of service offered in relation to the contribution of each user, must be the goal of contemporary P2P systems which aim at live streaming. The advantage of this technique is that it affects directly the quality of the video perceived and this way the users cannot help contributing. The emergency of the liveness of the emission constitutes a supplementary motivation. Of course such a technique is based on the existence of a scoring system, which can be determined in a way to ameliorate the functionality and the performance of the system.

2.3 Challenges

The nature of live streaming, and especially its sensitivity to delays, is the cause for many of the problems encountered when dealing with such an application, problems which would be less important in a Video On Demand system, where there is no notion of liveness and users are generally more linient, and almost inexistent in a content sharing application, where media file reproduction is done after downloading, so packet delays are not really an issue, as long as there is a satisfactory download rate. Some of these challenges are the following:

- Data nature: Live streaming is based on play-while-you-receive, that means that a sequence of pieces must be reproduced by the media player immediately at the reception or after a predetermined bufferisation delay. The latter one becomes a problem and must be of very limited duration in the case of video conference, where a considerable bufferisation delay may harm the interactivity of the communication. What's more the total duration of a media session is unknown, the data flow is generated dynamically and there is no prior knowledge of the content. The consequences of these two properties of live streaming are a very limited tolerance to packet delay and a rather high sensitivity to packet loss. Since a big safety margin doesn't exist (the bufferisation delay is equal to some seconds), a series of packets being lost or arriving late will very much harm the video quality, they may even cause an interruption, which is highly undesirable.
- Network conditions: In order for the P2P network to run properly and for the incentive mechanisms to be deployed with a predictable result, the network must present some special properties: Continuation of service even after sudden departures or failures, capacity to download simultaneously from different sources, ability to explore the network and find peers with interesting content and finally capacity to do a close to reality estimation of its own upload bandwidth and also of that of its neighbours.
- Network characteristics: In order to optimise the performance of any P2P system, we should supply it with the means to form its architecture taking into account the underlying internet topology, allow for a tcp friendly flow control in order to avoid congestioning the network with P2P traffic and finally take into account the fact that the majority of users have asymmetric internet connections (ADSL) in order to optimise the use of their available resources.
- Security and equity: This aspect of the encountered challenges concerns mostly the peers behaviour and habits. To protect the system from being exploited, we should introduce counter measures to defend to potential attackers or preferably minimise the probability of success of a possible attack by establishing appropriate rules and policies. The system should also be able to provide a minimal level of integrity of the exchanged data and most importantly a minimal level of fairness among the users.

3. Experiments – Experience with existing algorithms

3.1 Simulation Parameters

For user outbound and inbound bandwidth, I adopt the measurement results derived form actual Gnutella nodes in 2006.

Type	Inbound(kbps)	Outbound(kbps)	Fraction
DSL/Cable	784	128	0.2 among DSL/Cable
DSL/Cable	1500	384	0.5 among DSL/Cable
DSL/Cable	3000	1000	0.3 among DSL/Cable
Ethernet	10000	5000	Altered(0~0.15 among all)

Table 12: "Optimizing the Throughput of Data-Driven Peer-to-Peer Streaming", Meng Zhang, Qian Zhang, Table IV, **Bandwidth Distribution in Gnutella network 2006**

Instead of measuring the inbound and outbound bandwidth in kbps, I choose to measure it in blocks per iteration, where each iteration is typically set to 1 second. Since a common block size is 15 kbytes and my simulation includes 15 nodes, a new table is created. For the first simulations, no Ethernet node is supposed to exist. He will be introduced later on to deduce interesting results.

Type	Inbound (blocks/s)	Outbound (blocks/s)	Number of nodes
DSL/Cable	6	1	4
DSL/Cable	12	3	7
DSL/Cable	24	8	4
Ethernet	84	41	0 or 1

Table 13: Bandwidth Distribution in Simulation

If we suppose that the streaming rate is 300 kbps, we want to measure it in blocks/sec, so we have 300/8*15=2,5. I consider 3 blocks/sec. So the interest window of each node will slide 3 blocks/sec and its length is considered to be 100 blocks.
Since the goal is to study different scenarios for nodes with different resources, we suppose that for each node category (slow, medium, fast) there are four buffer states (10% full, 30% full, 50% full, 70% full) to simulate nodes who have arrived at different time instants. There is also a medium node that has just logged in, one with an almost full buffer and one with a randomly filled buffer. All this information can be easily seen in the following table. The buffers are filled randomly following a uniform distribution.

Type	Node ID	Inbound (blocks/s)	Outbound (blocks/s)	Initial Buffer State (buffer size 100)
Slow	0	6	1	10
Slow	1	6	1	30
Slow	2	6	1	50
Slow	3	6	1	70
Medium	4	12	3	0
Medium	5	12	3	10
Medium	6	12	3	30
Medium	7	12	3	50
Medium	8	12	3	70
Medium	9	12	3	90
Medium	10	12	3	Random
Fast	11	24	8	10

fast	12	24	8	30
Fast	13	24	8	50
fast	14	24	8	70

Table 14: Bandwidth and Buffer State Distribution in the Simulation

In these simulations, each node acquires packets through two mechanisms, first by his fellow 14 nodes by comparing their bitmap to his and second by a random mechanism, following a geometric like distribution, so that the blocks at the beginning of the interest window (left), which will soon be played, are obtained with a greater probability than blocks at the end (right). The goal of the second mechanism is to simulate the effect of a wider neighborhood, which means that each node has transactions with others outside the known group of the 15. Although we cannot simulate these nodes, this way we take into account their presence. In order to stick to a realistic scenario, we suppose that each node is only allowed to obtain by the random mechanism the number of blocks he has acquired by the regular mechanism plus 3 more blocks. We add these three blocks in order to avoid the case where a node becomes unable to find missing blocks owned by his peers and starves, if we don't allow him to get them by the random mechanism. Especially for node 4, there is no random mechanism deployed because we want to isolate and study his interaction with the 14 known nodes. The pseudo code of the main parts of the program for the simplest case scenario can be found in the Appendix at the end of this document.

3.2 Upload Capacity Based Selection

For each scenario, 25 simulations are launched, all with the same initial configuration in order to have a statistic result and minimize the effects of bad thread handling by the java machine. The results are illustrated in the following figures.

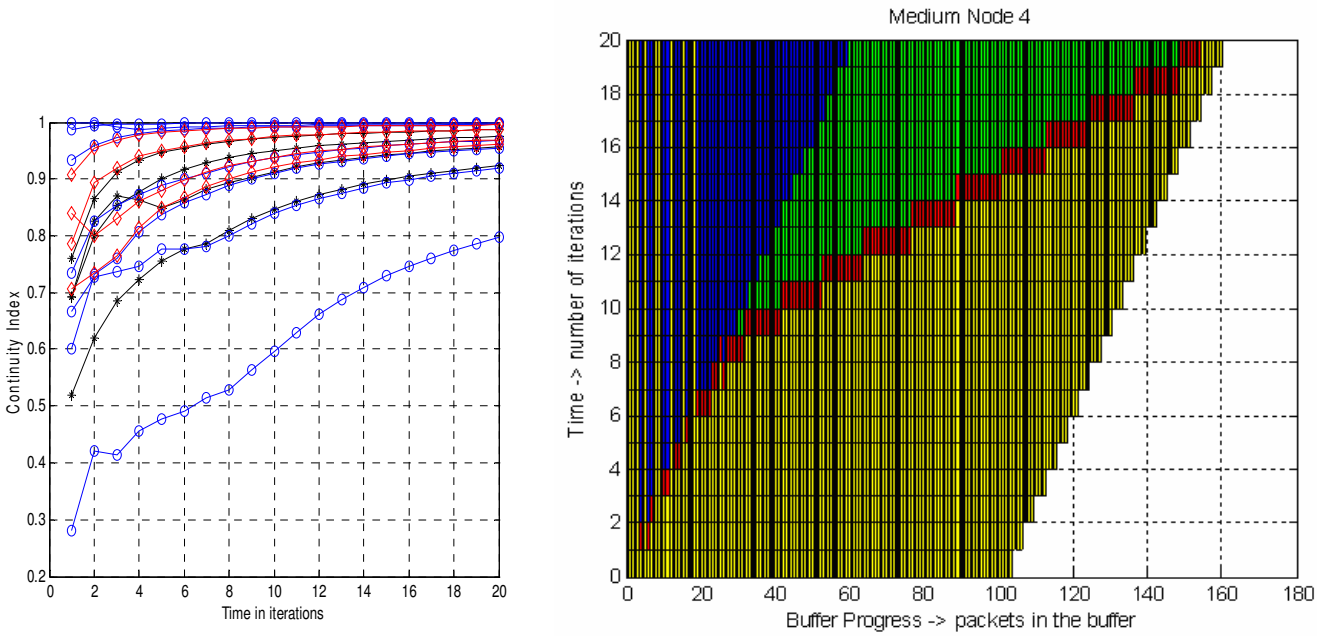


Figure 6: Upload Capacity Based Selection, left Continuity Index, right Buffer Progress of node 4

In the left part of figure 6, the x axis represents the time evolution in iterations (total 20 iterations) whereas the y axis represents the continuity index of each node. In consequence, each dotted point is the continuity index of a node at the end of the particular iteration. The continuity index is a very common measure of the quality of service in live streaming applications, in particular it is the percentage of packets who

arrived before their play out time. The rest are packets who either never arrived either they arrived late and subsequently they were rejected.
In order to derive interesting conclusions, the different node categories are painted in different colors in the preceding figure. Slow nodes are painted in black, medium nodes in blue and fast nodes in red. I remind that for each node category, there are four possible initial buffer states (10%, 30%, 50% and 70% full). The lowest blue line represents the evolution of the continuity index of node 4, who is a medium node who starts with an empty buffer. We can see that he converges slowly but his performance is much worse than that of his partners. In this scenario, the empty initial buffer constitutes a serious handicap. If we don't look at node 4, the lowest line of each color always concerns the node of that category who starts with a 10% full buffer and each superior line concerns a superior buffer state. If we compare the first black, blue and red line, it is obvious that the fast node who starts with an empty buffer is much more successful in retrieving quickly the missing pieces than the slow and the medium one. What's more, it is evident that fast nodes converge faster to a higher continuity index value than medium ones. In the end, we should note that the two nodes (9 and 10) who start with an almost full buffer, present a continuity index constantly equal to 1 (the two lines on top).

The next figure represents the mean value of the final continuity indexes at the end of each simulation and allows us to compare the performances of the different node categories.

Node	Initial Buffer State (%)	Continuity Index
0	**10**	**0.9233**
1	**30**	**0.9567**
2	**50**	**0.9753**
3	**70**	**0.9867**
4	**0**	**0.7967**
5	**10**	**0.9193**
6	**30**	**0.9553**
7	**50**	**0.9973**
8	**70**	**0.9960**
9	**90**	**0.9993**
10	**Random**	**0.9687**
11	**10**	**0.9607**
12	**30**	**0.9693**
13	**50**	**0.9873**
14	**70**	**0.9953**

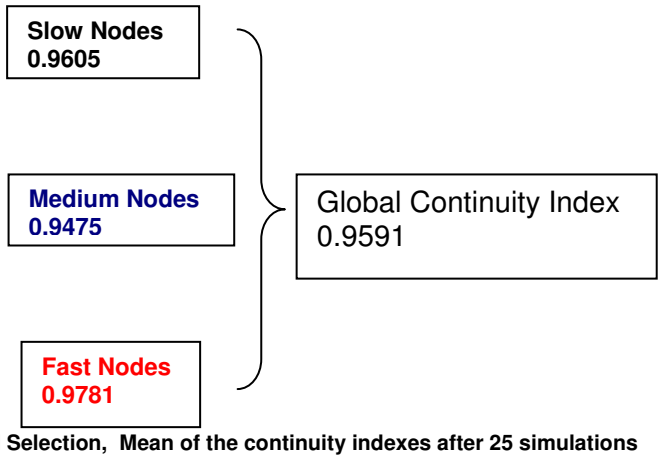


Figure 7: Upload Capacity Based Selection, Mean of the continuity indexes after 25 simulations

One thing that strikes as odd at this figure is that the overall performance of the medium nodes is slightly worse than that of slow nodes. The main reason for this phenomenon is the existence of node 4 who starts with an empty buffer and is seriously handicapped as we have already explained. However, nodes 5 and 6 present also slightly worse performance than the nodes 0 and 1, although they begin with the same buffer state. The logical explanation for this, is the following: Since the buffers are filled randomly following a uniform distribution, it is possible that one buffer with 10 initial packets is more beneficial than another one, because these 10 packets are more concentrated at the beginning of the buffer, which is its most crucial part since the node won't have many chances to fill it.

Now, in the right side of figure 6, we can see the evolution of the buffer of the medium node 4 who starts with an empty buffer initially and whom we use as a case scenario of a new comer with medium resources. The figure represents the mean of the buffer states of the consecutive 25 simulations for node 4. That means that for each packet of the buffer for each iteration of each one of the 25 simulations, we

register if it has been acquired or not. In the end, if a packet has been acquired in more than 12 simulations, we consider that it exists in the final figure. If not, we leave its location blank. So, the x axis represents the pieces that have been acquired and exist in the buffer and the y axis the 20 consecutive iterations. The blocks on the left are the first to be played by the media player. We must also note that since we simulate a case of live streaming, the interest window slides to the right, 3 blocks at each iteration. All this information is illustrated by the different colors in figure 6 right. The blue edge represents the current position of the media player, which means that all the blue packets have already been reproduced. The yellow edge represents the current position of the interest window, the yellow blocks are the ones the peer has not obtained yet and he is interested in. The green blocks are pieces that have been acquired but haven't been played yet and the red blocks are the new pieces that have been obtained during the last iteration.

After explaining the meaning of the colors, we can now explore more interesting aspects of this figure. It is obvious that at the beginning of the simulation, node 4 isn't able to obtain not even the three necessary blocks per iteration to maintain a smooth video quality. The reason for this is the existence of a race condition among the nodes, half of which are new comers (7 out of 15 nodes have less than 30 blocks in their buffers). So, the available resources are considerably monopolized by the faster nodes, who present a high continuity ratio even at the beginning, and the other nodes are condemned to starvation. After a while, though, faster and older nodes fill their interest window, so there is place for the new comers, who begin to ameliorate their continuity index and acquire more new packets (red) at each iteration. This situation continues and at the end of the 20th simulation, node 4 has almost filled his interest window.

We also conducted the same simulation for a longer duration (60 instead of 20 iterations) and we observed that the evolution of the figures is normal and predictable. These figures can be found in the Appendix at the end of the document.

After this example, we can clearly see that fast nodes are naturally busted by their high capacities and resources at the expense of slower nodes. So the need for incentive mechanisms, which could guarantee a kind of fair scheduling among all the participants, become evident, in order to help all the peers obtain the stream on time and with a good quality. The clear unfairness in service, that we have just observed, justifies this whole work and the seek of a "remedy" to the problem is essentially its goal.

3.3 Bitmap Resemblance Based Selection

Until now we examined the case where the only criterion to select one's neighbors was their upload capacity. But what if the next uploader isn't selected depending on his upload rate but on his bitmap resemblance to ours? The next case scenario examines exactly this situation. What happens is the following: Before each download demand, each node finds the positions of the first five non existing blocks in his own bitmap. Afterwards he searches these five positions in all the remaining bitmaps of the other nodes and chooses the node with the least resemblance bitmap to his. That is, in the best case he finds a node who owns all the five missing blocks, if there isn't such a node he chooses one with 4 missing blocks and so on. If there are multiple nodes with the same resemblance ratio, he chooses the one with the highest upload rate. We can see the results of this policy in the next figures and compare them with the respective results of the first simpler scenario. The goal is to observe which of the two criteria is most important and powerful, in order to know which one to take into account or how to combine them in a potential more complicated incentive mechanism. We also would like to see how the different node categories react in these two cases, so as to find possible vulnerabilities.

In figures 8 and 9, we can observe that the performance here is slightly worse than in the first case in terms of continuity index (global continuity index 0.9366 in comparison to 0.9591 in the first case). What's more, a very interesting phenomenon is that the performance of the faster nodes is affected more than the performance of the slower ones. If we compare the cumulative continuity indexes of the three categories, we can see that that of the faster nodes deteriorates more than that of the slower ones. Why does that happen? Because now, faster nodes may choose a slow or medium node as uploader thanks to his bitmap resemblance, whose the upload capacity though is very limited compared to their download capacity. For example, they may choose a slow node who owns all the five missing blocks they demand without considering that this particular node is only capable of serving them one. In conclusion, we can see that by all means this node selection strategy stalls the faster nodes without really ameliorating the performance of the slower ones, whose continuity indexes remain almost the same. These conclusions are reinforced by the right part of figure 8, where we can see the buffer evolution of node 4, whose continuity index is much worse now than in the first case.

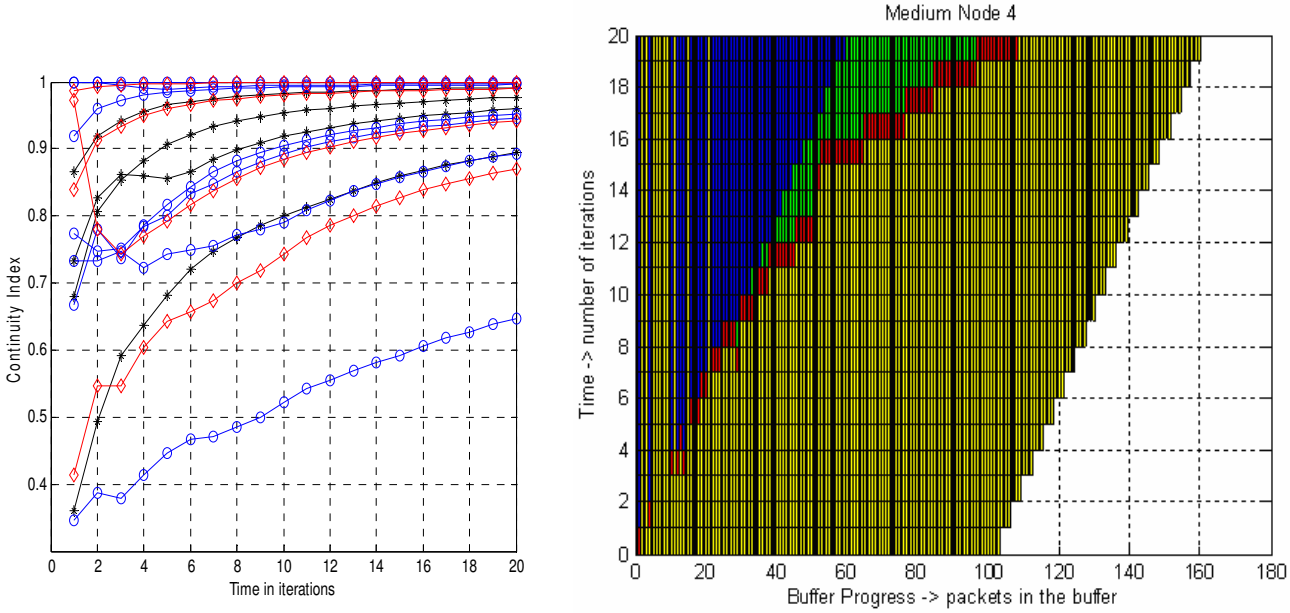


Figure 8: Bitmap Resemblance Based Selection, left Continuity Index, right Buffer Progress of node4

Node	Initial Buffer State (%)	Continuity Index
0	10	0.8947
1	30	0.9593
2	50	0.9767
3	70	0.9913
4	0	0.6467
5	10	0.893
6	30	0.9527
7	50	0.9973
8	70	0.9960
9	90	1
10	Random	0.9460
11	10	0.8707
12	30	0.9420
13	50	0.9900
14	70	0.9993

Slow Nodes 0.9555

Medium Nodes 0.9188

Fast Nodes 0.9489

Global Continuity Index 0.9366

Figure 9 : Bitmap Resemblance Based Selection, Mean of the continuity indexes after 25 simulations

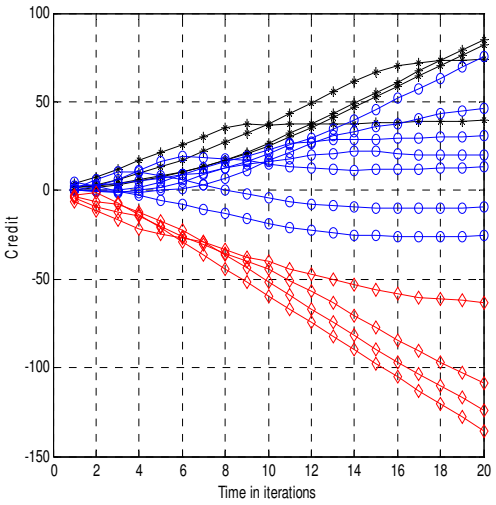


Figure 10: Credit of the 15 nodes, 3.3

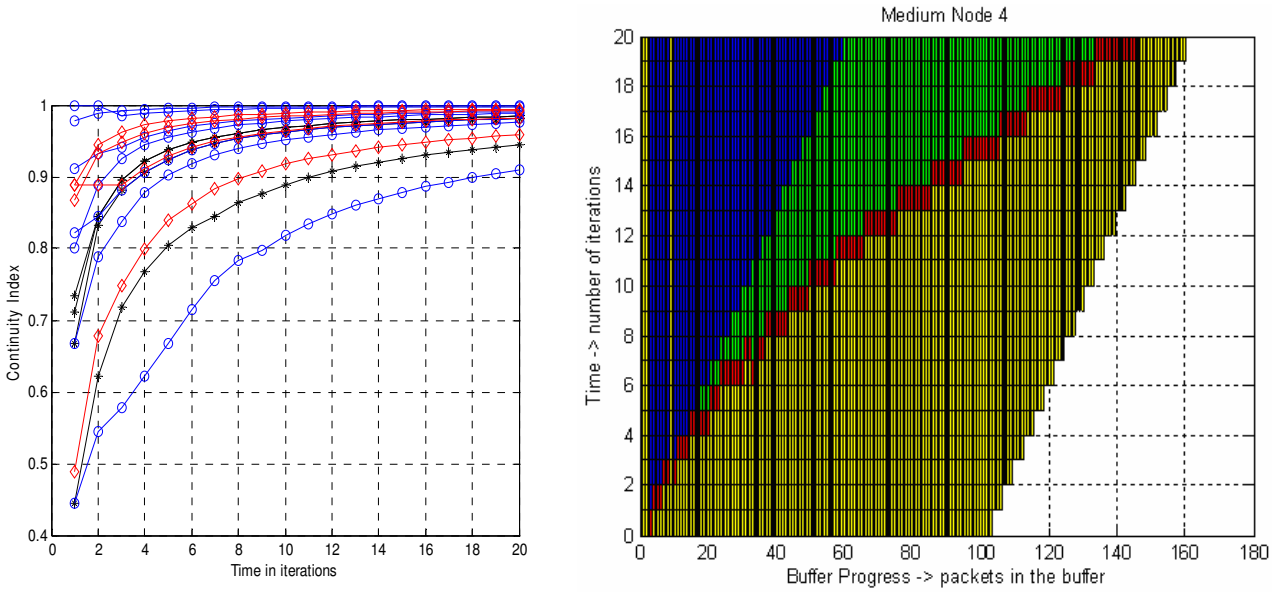
In this figure we have represented the credit of each node. In order to do that, we have supposed that each node begins with zero credit, he gains a point for every block he downloads and he looses a point for every block he uploads. So in total the credit of every node represents his total downloads minus his total uploads. It is obvious that fast nodes upload much more than they download and slow nodes the opposite, they download much more than they upload. Medium nodes are somewhere in the middle. From this figure, an already known fact becomes certain, faster and richer nodes are the systems' main contributors and the ones that keep it running by serving the slower ones. All that considering that all the nodes are altruistic, there is no selfish behavior. This possibility will be examined later on.

3.4 BitTorrent Choke Algorithm Upload Only

After having examined the results of the two previous cases, where the selection of the next uploader was done based on quite simple criteria, upload capacity on the first case, bitmap resemblance on the second one, it is time to study a more complicated scenario, the BitTorrent choke algorithm. In order to test thoroughly its functionality, we decided to divide the mechanism in two phases and study them separately. First we simulate a policy where at each unchoke period, each node discards the worst of his 4 uploaders and replaces it with another node at random. Normally, a full choke algorithm would also follow the same procedure with the downloaders, which is simulated in the next case and it constitutes the second phase. The goal of this step by step study of the algorithm it to observe the impact on efficiency and performance.

So, as we can see in the left part of figure 11, this policy seems to perform significantly better as all the nodes' continuity indexes converge to a value over 0.9. In fact, 13 out of 15 nodes have a continuity index over 0.95 and 11 out of 15 over 0.98, which is impressive and guarantees an exceptional video quality.

Figure 11: BitTorrent Choke Algorithm Upload Only, left Continuity Index, right Buffer Progress of node 4



These conclusions are reinforced by the following table, which summarizes the continuity indexes of the 15 nodes at the end of the simulation.
In the right part of figure 11, we can also observe the evolution of the buffer of node 4. We can see that the quantity of data obtained at each iteration is much more regular now, which allows node 4 to achieve the desirable performance.

Node	Initial Buffer State (%)	Continuity Index
0	**10**	**0.9444**
1	**30**	**0.9811**
2	**50**	**0.9844**
3	**70**	**0.9844**
4	**0**	**0.9089**
5	**10**	**0.9811**
6	**30**	**0.9756**
7	**50**	**0.9978**
8	**70**	**0.9889**
9	**90**	**0.9989**
10	**Random**	**0.9911**
11	**10**	**0.9589**
12	**30**	**0.9822**
13	**50**	**0.9922**
14	**70**	**0.9944**

Slow Nodes 0.9736

Medium Nodes 0.9775

Fast Nodes 0.9819

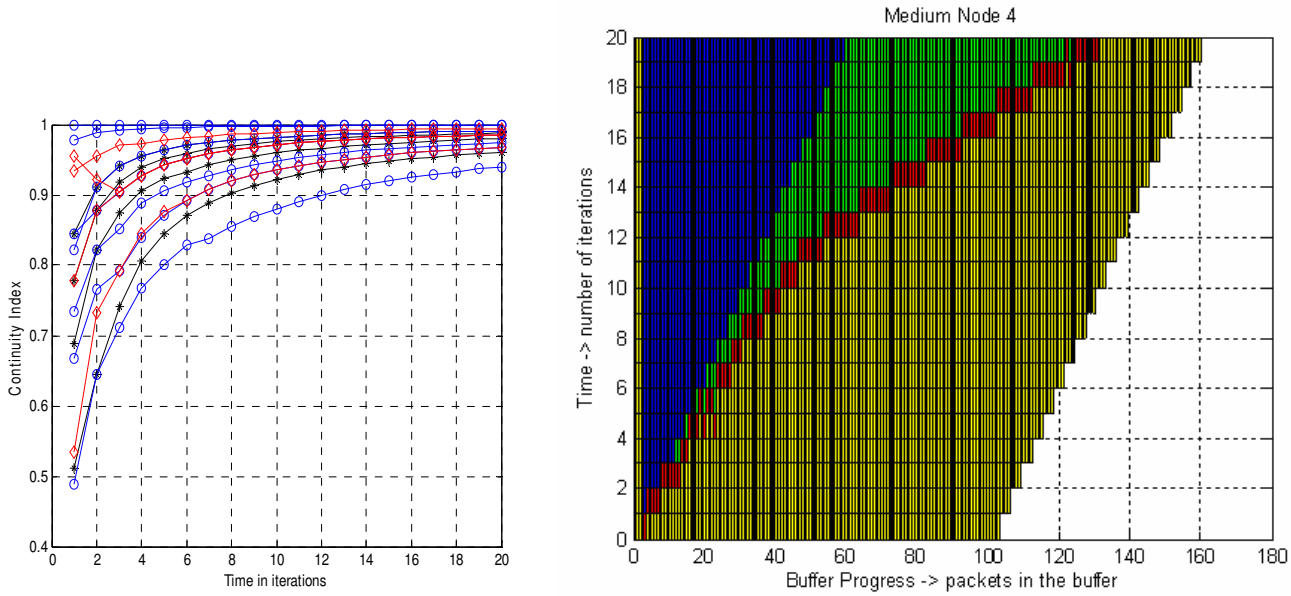
Global Continuity Index 0.9776

Figure 12: BitTorrent Choke Algorithm Upload Only, Mean of the continuity indexes after 25 sims

3.5 BitTorrent Choke Algorithm Upload – Download

This case simulates a full choke algorithm: At each unchoke period, each node discards the worst of his 4 uploaders and replaces it with another node at random. He also does the same with his worst downloader, replacing it with another one at random.
So, as we can see in figure 13 left, this policy seems to perform even better than the last one with 14 out of 15 nodes finishing with a continuity index over 0.95. Only, node 4 who starts with an empty buffer converges below 0.95, at 0.94. As a conclusion, it is obvious that the BitTorrent choke algorithm works impressively by allowing all the nodes to make the most of the available resources.

Figure 13: BitTorrent Choke Algorithm Upload – Download, left Continuity Index, right Buffer Progress of n4



Node	Initial Buffer State (%)	Continuity Index
0	10	0.9611
1	30	0.9800
2	50	0.9911
3	70	0.9878
4	0	0.9400
5	10	0.9744
6	30	0.9678
7	50	1
8	70	0.9911
9	90	0.9989
10	Random	0.9856
11	10	0.9678
12	30	0.9856
13	50	0.9856
14	70	0.9944

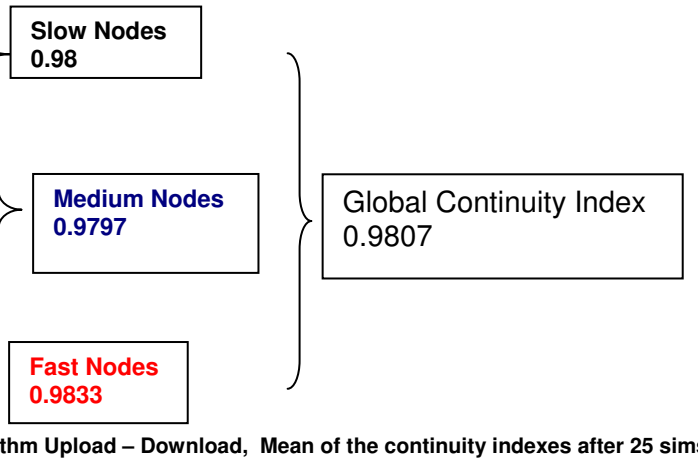


Figure 14: BitTorrent Choke Algorithm Upload – Download, Mean of the continuity indexes after 25 sims

The right part of figure 14 is quite significant. We can see that at the beginning of the simulation, node 4 isn't able to obtain many pieces at once because of the choke algorithm and the fact that he hasn't got anything to share, so he is rejected by the majority of the nodes, and he is mostly counting on being randomly selected as a partner, in order to download the necessary pieces. After the first 6-7 iterations though, he has enough pieces to share and we can see that the quantities he obtains at each iteration augment.

So, if the BitTorrent choke algorithm is so efficient, why continue this study and not just adopt it as the best solution? It is true that this mechanism is very reliable but it has a major flaw: It is very lenient with free riders, who can download a file by taking advantage of the optimistic unchokes, surely more slowly than a node with a good behavior but still they can do it. So, BitTorrent is not an optimum even for file-sharing applications. However in file sharing there are no delay constraints, so a potential free rider doesn't really care if the download duration is 3 hours instead of 1, which is not the case in live streaming, where long periods of waiting before or during the streaming are unacceptable.
What's more, there is a more crucial reason for which BitTorrent choke algorithm is inappropriate for live streaming. In a file sharing application, there is always a considerable amount of seeds, nodes who have already downloaded and own all the pieces of a particular file. In live streaming, only the source owns the totality of the file and it surely cannot serve all the peers demanding it, so the need to motivate with proper incentives all the users to contribute and cooperate continuously is far more vital to keep the system alive and running.
As a conclusion, the aim of this work is to introduce stronger incentives for contribution and stricter consequences for misbehavior. We show an example of such a mechanism in the next section.

4. Study of Incentive Mechanisms for Live Streaming

4.1 Differentiated Services

In this next simulation, a new incentive strategy is tested, which is based on a differentiated services policy supported by a reputation system. To begin with, we must note that the proposed solution concerns an unstructured and decentralized P2P network. This means that there is no central entity in the form of a server, or a super peer of even a torrent, in order to avoid the risk of a potential bottleneck or single point of failure. What's more, we suppose that every user receives the stream by many peers simultaneously, so it's a case of a data driven architecture, like the one used in DONet. To conclude this short introduction, it must be mentioned that the use of time shifting or video patching techniques would significantly ameliorate the perceived video quality and eliminate the negative effects of connection interruptions and sudden peer departures.
The main idea of this kind of techniques is that each and every user is only allowed to download data from nodes inferior or equal to him. The position of each node in the global distribution, which determines its relation to its peers (inferior, superior, equal), is given by a scoring function based on a reputation system. The essential problem of this solution is the fact that the nodes must calculate their own scores since there is no central entity to play this role. However, all the recent studies of the users' behavior prove that they tend to be selfish and act strategically in order to ameliorate their performance, so it would be logical to suppose that a great majority of them would be prone to lying concerning their contribution. So, the proposed solution should also take this into account and try to discourage this kind of behaviour.
To begin with, I will describe the scoring function. Each user maintains a short history of his transactions with all the other pairs and every time he has to, he calculates his opinion concerning another node based on this history. This opinion may concern the quantity of the served data, the available bandwidth of the node, the service duration, the propagation of the node's demands and a number of other representative characteristics of a user's behaviour. In our case, we use a simple scoring function, which considers only the upload capacity of a certain node and his contribution to us. This way a score between 0 and 1 is attributed to each one of a node's peers.

$$score[i] = 0.5\frac{upload\_capacity[i]}{\max imum\ \ upload\ \ capacity} + 0.5\frac{contribution\_to\_us[i]}{\max imum\ \ contribution\ \ to\ \ us}$$

Figure 15: Scoring Function used in simulation

In this simulation, each node has to store information concerning his 14 fellow nodes of the simulation only. In a real world scenario, the number of nodes a user "meets" during a session is potentially very large, so it is impractical and impossible to take everything into account, a part of the available information must be excluded or deleted after a period of time so as not to monopolize a node's memory space. This means that the history kept by each user must have a limited size, so it must refer to a limited period of time, that is to its most recent transactions with his environment. This way, each user is judged and scored according to his most recent actions and in consequence he can't take profit of his good reputation to exploit the system, at least not for long. However, this mechanism poses great difficulties for the new comers, since they have nothing to share, in order to augment their score, and it is a bit unfair with nodes who present regularly good unselfish behavior, since it doesn't remunerate them for it. On the other hand, a total presence of a node over a large period of time is useless for a live streaming application, where it is urgent that all the

nodes contribute regularly all the time, so this characteristic is not necessarily negative. In any case, this simulation concerns a small neighborhood of peers over a small period of time, so none of these mechanisms is implemented, and the final score of each peer is a cumulative measure of his behavior during all the duration of the simulation. We also suppose, for the time being, that there are no strategic nodes who try to trick the system.
So, after describing how the scores are determined, it is time to explain how the differentiated services system works. When a transaction starts, that means when a node demands service or when a node receives a demand for service, the two parties calculate their opinion of the other using the following function.

$$opinion = 0.5 * score[i] + 0.5 \frac{opinion\_of\_others[i]}{number\_of\_others}$$

Figure 16: Opinion calculation function used in the simulation

At this point, it is important to note that with the preceding function, we introduce a kind of global history of the system, as each node calculates his opinion for everybody else, depending partly on his own information and party on the information given by his neighbors. Of course, his own opinion is more important so it participates with a higher factor (0.5) than those of the others (0.5/14=0.0357), but in any case this technique introduces a minimum of objectivity and fairness. This way, at the end of this phase, each one of the two nodes has calculated the score of the other and they exchange these two values. If the score of the node that demands service is inferior, the other node has the liberty to reject him, but if it is superior or equal, he is obliged to serve him. The advantage of this procedure is that none of the nodes calculates or even knows his own score, so it is more difficult to lie and cheat.
At this point, we can introduce a number of different measures to improve the system, some of which are the following:

1) If the node with the inferior score, who is obliged to serve his peer, refuses to do so, the demanding peer propagates a message informing the other nodes of his misbehavior in order to penalize this node by diminishing their opinion of him (decreasing the existing score that concerns him).
2) We can introduce a special treatment for the new comers, as for the time being they are confronted as selfish non-contributing nodes, and they are severely penalized, as we will see in practice. We could for example, consider that when none of the nodes has a stored score for a certain node, the latter is a new comer and he must be served anyway and by all means.
3) We could also elaborate the existing mechanism by issuing an inverse procedure to the one propagating a message of foul play when a node refuses to co-operate. This procedure would concern the altruistic nodes who serve their fellow peers, even though they have a superior score and so they are not obliged to do so. At this case, the served node propagates a message of “praise” to inform the others of the virtues of his partner and in consequence remunerate him by proposing to everyone to augment his score.
4) Finally, we could also introduce intelligent algorithms to discover and exclude “liars” and impostors. Although there are quite a lot of mechanisms proposed by the existing literature, the overhead they add is considerable and the probabilities of false positives and false negatives are not to be ignored.

As I have already remarked, to begin with a simple scenario, we suppose that all the nodes follow the existing rules, so these 4 propositions are not implemented in the simulation to follow.

4.1.1 Differentiated Services - Simple Case

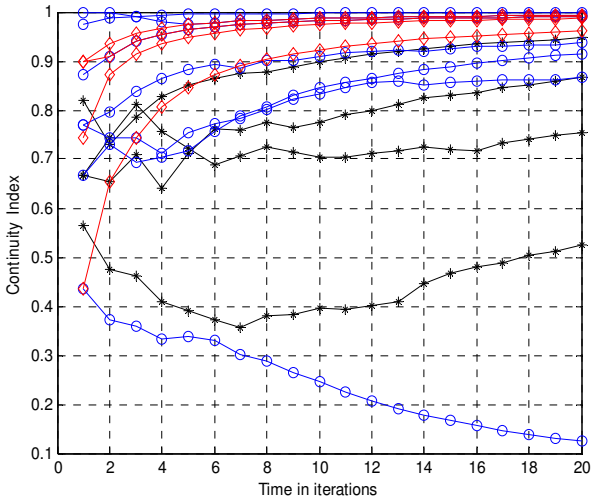


Figure 17: Differentiated Services – Simple Case, Con. Indexes

In figure 17, we can make some very interesting observations. Nodes 0 to 6, which are all the slow nodes and the medium nodes who begin with a poor initial buffer (node 4 – empty, node 5 – 10% full, node 6 – 30% full), tend to present a decrease in their continuity index before stabilizing their performance and converging to a higher value. The reason this happens to these categories of nodes is the fact that they can't obtain a high score, which would allow them to download data from their peers, either because of their low upload capacity either because of their empty buffer which is of no interest to the others. So, they need some time in order to obtain enough blocks to be able to raise their score with constant contribution, without however for most of them succeeding in reaching a satisfactory value (5 out of 7 are below 0.9, which is hardly a quality limit). This situation becomes a sad truth especially for node 4, who begins with an empty buffer and unfortunately finishes in almost the same situation, as we can see in figure 20. To understand better this figure, we must consider that at the beginning almost every node has a particularly low score, except maybe the 4 fast nodes, so node 4 succeeds in acquiring a few pieces by his neighbors. After a while though and because of the fact that the pieces obtained have already been played and are no longer wanted by the other peers, node 4 is technically incompetent of doing anything in this system. This is a major flaw of this strategy, since as we have seen, new comers and slow nodes are severely handicapped and we would naturally expect them to disconnect sooner or later. The reason is that our differentiating policy is extremely harsh and strict. As a result, slow nodes and new comers can't benefit of the spare capacity, although it exists and it is lost. Obviously, we must reconsider our notion of equity and allow for a more flexible bandwidth allocation to nodes with lower contribution rates.

4.1.2 Differentiated Services with Mechanism to help new comers

A logical decision to make is to permit the poor nodes, in terms of pieces or upload capacity, to use the available bandwidth after the rich nodes have finished with it. In order to do so we use the following strategy: We introduce a global counter which has a value equal to the sum of the upload capacities of all the known nodes. Every time a piece is downloaded by a node, this counter is diminished by one. So, we can logically suppose that after a few iterations, when the fast nodes have almost filled their buffers and the slower ones haven't got a score high enough to download, at the end of each iteration or exchange period this counter will have a non zero constant value. At this point, slower nodes are allowed to download data even if they don't deserve it, in terms of implementation they are gradually allowed to automatically augment their score in order to have access to superior peers. The figures showing how the global behavior of the system changes follow.

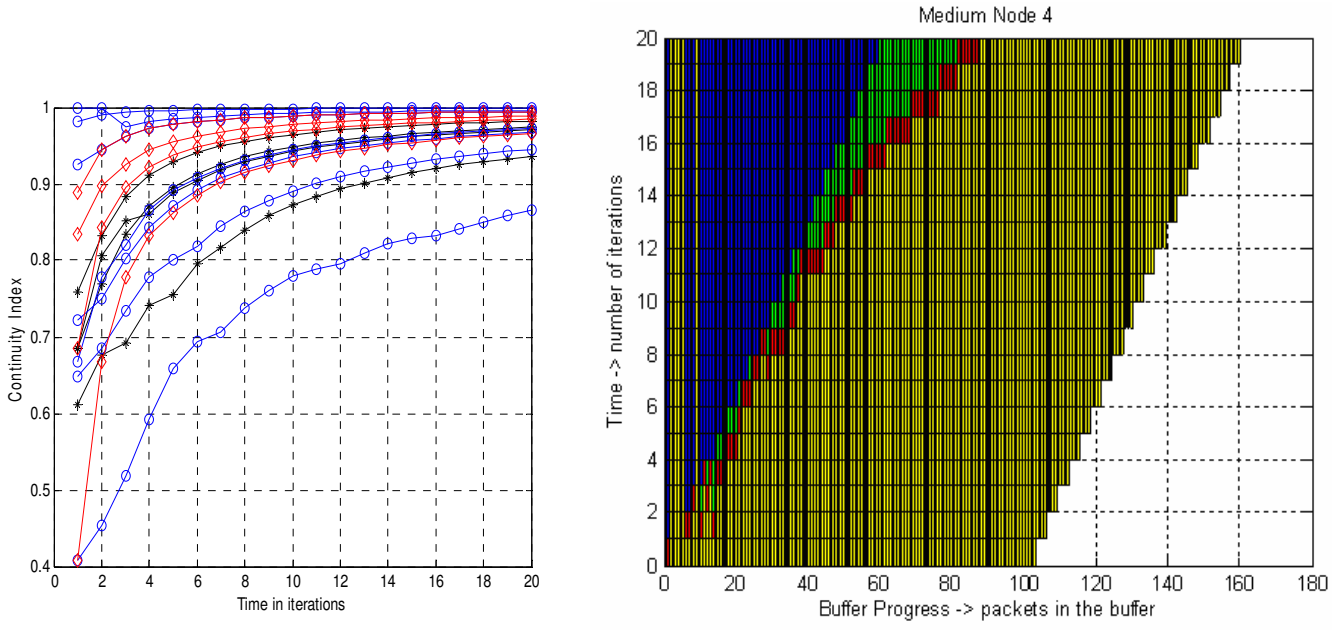


Figure 18: Differentiated Services with Mechanism to help new comers, left Continuity Index, right Buffer Progress of node4

0	10	0.9361
1	30	0.9713
2	50	0.9741
3	70	0.9824
4	0	0.8657
5	10	0.9454
6	30	0.9722
7	50	0.9963
8	70	0.9944
9	90	0.9991
10	Random	0.9676
11	10	0.9657
12	30	0.9889
13	50	0.9843
14	70	0.9944

Slow Nodes
0.9660

Medium Nodes
0.9630

Fast Nodes
0.9833

Global Continuity Index
0.9692

Figure 19: Differentiated Services with Mechanism to help new comers, Mean of the continuity indexes after 25 simulations

In the left part of figure 18, we can see that none of the nodes presents a decrease in its continuity index before converging to a higher value and most important, even node 4 can now attain a point where its performance becomes satisfactory. In fact, the performance of the majority of the nodes is really high. If we compare it for example with figures 13 (BitTorrent choke algorithm), we can see that the continuity indexes in BitTorrent are slightly better (less than 0.2 better for slow and medium nodes, exactly the same for the fast nodes) except for node 4, which in our case is more penalized as a new comer and converges at a value around 0.87 after 20 iterations whereas in BitTorrent this value was 0.94.
So, we can easily see that the main vulnerability of the proposed algorithm is the way it confronts new comers. However, is this a real vulnerability or a sign of a greater resistance to free riders? In order to answer this question, we will study the effects of the presence of free riders in the proposed algorithm and in BitTorrent in order to draw conclusions and compare their respective performances.

4.1.3 Study of the Free Riders Factor

To do so, we alter the role of medium node 10 in order to imitate the behavior of a free rider. That means that node 10 begins with an empty buffer and he always advertises an empty bitmap even if he has actually obtained some blocks, because his intention is not to contribute at all. He is in a similar state as node 4 with the only difference that node 4 is honest about the contents of his buffer and he is willing to contribute if there are nodes interested in what he has to offer. In the following figure, we can see on the left the continuity index diagram for the BitTorrent case and on the right the same diagram for the proposed differentiated services paradigm.

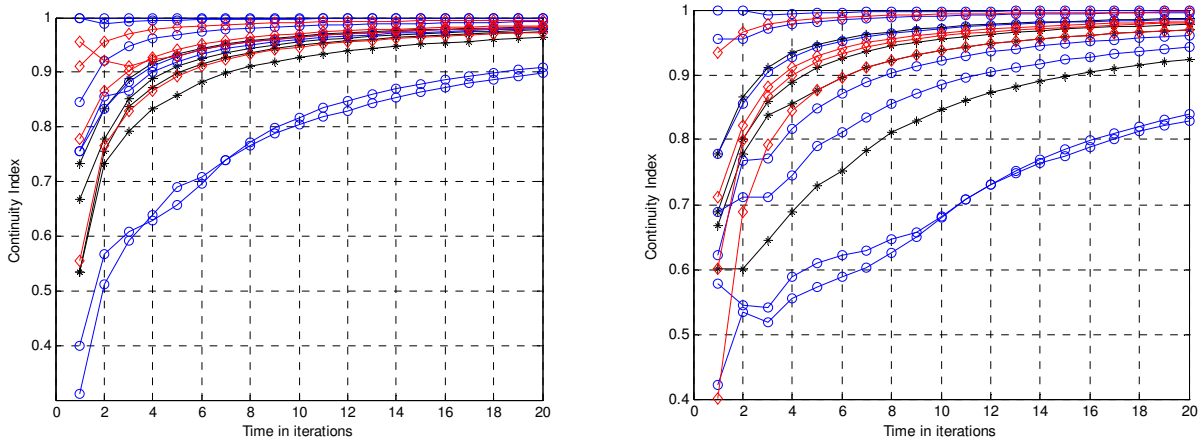


Figure 20: Comparative study of the continuity index diagrams for the BitTorrent and Differentiated Services

We can easily observe that both the algorithms are not able to make a distinction between an innocent new comer (node 4) and a malicious free rider (node 10), since their final continuity indexes are almost the same. However, the proposed mechanism penalizes both of them, as they are only able to attain a value around 0.83, whereas BitTorrent is less strict and allows them to reach a continuity index over 0.9. So, at this point we come face to face with a very common dilemma of incentive techniques and we are obliged to do a compromise: Do we prefer a mechanism with greater resistance to free riders who also penalizes the new comers or a more flexible one where free riders can quite easily exploit the system? Generally, the distinction among these two node categories, new comers and free riders, is a very difficult and complicated problem in P2P systems: The majority of free riders don't present a monolithic behavior and tend to imitate new comers in order to pass unnoticed, generally they will not reject every single demand to upload data but they will contribute as less as possible. So, in order to detect them, we should at least be able to keep a detailed and long history of the actions of every single node towards every one else, which is extremely costly in resources and practically impossible. Under the light of these observations, we think it is wiser to prefer a system, which is more skeptical towards nodes with low contribution, even at the expense of new comers, than a naïve one, which would help new comers on one hand but also get easily fooled by free riders on the other hand. The final choice depends on the exact properties of the system we want to build.

4.2 Micro Payment Example with Bufferisation Delay

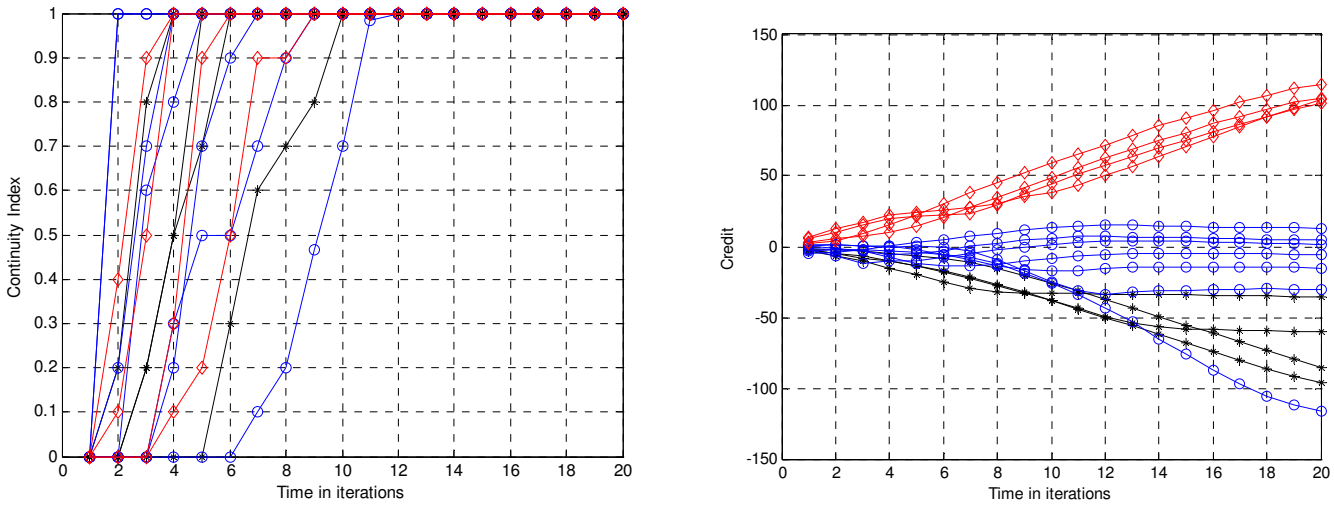
At this point, it would be interesting to change a little the initial configuration in order to explore different aspects of the system's behavior and come to a better understanding of the way the parameters interact to produce the final result. The changes are summarized in the following paragraphs:

1) Until now the initial buffers of the nodes were filled following a uniform distribution, that is completely at random. Although this introduces a higher level of difficulty to the system and guarantees that the results are not affected by a potential favorable distribution, it would be more realistic to fill the buffers following a geometric like distribution, like the one of the random mechanism. It is more logical, since nodes that have been in the system longer are more likely to have demanded and obtained blocks which are located at the beginning of the buffer than at its end. A rational supposition is that this change will probably ameliorate the system's performance, without however disturbing the relative position of each node in the global distribution or the relative distances among them in the final performance metrics.
2) Until now, we considered that the reproduction of the live stream begins as soon as the node gets connected, that means that there is no bufferisation delay which would also constitute a safety margin and would much improve the final system performance. So, after having examined the system's behavior under these difficult conditions, we will now introduce a bufferisation delay of 21 pieces (the size of the interest window is 100 pieces). This means that the reproduction of the stream doesn't begin unless the 21 first pieces have bee acquired, so there is no point now in measuring the continuity index from the beginning of the simulation, we should rather start counting from the instant the play out starts. What 's more, it is very likely now to observe very high continuity indexes, since there is a really large safety margin of 21 packets, that means that a node has the safety to obtain nothing for 7 consecutive iterations, which is highly unlikely, and he still won 't notice any negative effects on the video quality. In fact, the continuity index doesn't seem to be a really representative quality measure any more, we would suppose that it would converge very quickly to a value around 1 and stay invariant, unless something like a flash crowd or a number of massive sudden departures happens. So, we introduce a new measure, which seems to have more meaning, and it is no other than the iteration at which the play out begins, the instant at which the node obtains the 21 first pieces.

4.2.1 Simple Case

So, now let's see the impact of these changes on the first case we examined, which is the simplest scenario, where each node selects to download from the node with the highest upload rate.

Figure 21: Simple Case with Bufferisation Delay, left Continuity Index, right Credit



Concerning the left part of figure 21, we have manually verified that in all the simulations, with only two exceptions for node 4, the continuity index remains at zero until the node obtains the first 21 pieces and then jumps and remains constantly at 1. So, obviously we should perceive the preceding figure differently now. The evolution of each line is a joined measure of the probability that the play out starts at each specific iteration and of the mean continuity index at that iteration. Let's observe for example the right most line, which corresponds to the medium node 4 who starts with an empty buffer. The point (8,0.2) means that for 20% of the simulations done, the play out starts after iteration 8. What seems as a right quality measure is the point of the x axis which corresponds to a percentage of 0.5, which is the iteration after which the play out starts for half the simulations. In our case this must be around 9.2, which means that for half the simulations the play out starts after iteration 9.2. It is meaningful to accompany the previous figure with the next table, which represents exactly what we have just explained, the mean iteration at which the play out begins. The reason figure 21 left is important is so that we keep in mind that the instant when the play out starts is a mean and not an exact number and also that in some cases there is a considerable variance, like in the case of node 4. However, even in that case we can observe that there are less than 20% of the simulations with a play out time earlier than the 8^{th} iteration and less than 20% with a play out time later than the 10^{th} iteration. So the mean represents a considerable majority, it is not meaningless, although sometimes the appearance of a rather large values dispersion for a node or two seems bizarre. We have noticed though that nodes poor in resources (either with poor initial buffer or with low bandwidth) tend to present a greater variance than rich ones, which signifies that their behavior is less predictable. With this information we can easily calculate the mean play out start iteration of the system which is **3.5467**.

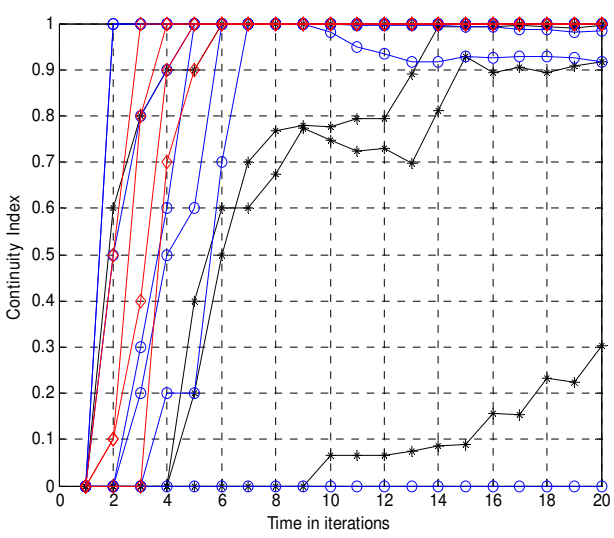
Node	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Mean play out iter.	6.6	3.6	3.3	2	8.4	5.1	4.2	2.6	1	1	2.1	5.4	3.8	2.4	1.7

Table 15: Mean play out start iteration for the 15 nodes after 10 simulations

In the right part of figure 21, we can see how each node's credit varies with time, if we attribute a zero score at the beginning to each one and continue by adding a point for each uploaded block and subtract one for each downloaded block. For the time being, the scores don't constitute a limit, we haven't introduced an incentive mechanism based on them yet. The point of this short comment is to verify what we have already seen, that fast nodes are the major contributors and slow nodes the major consumers, with medium nodes located somewhere in the middle.

4.2.2 Introduction of the Micro Payment Mechanism

Figure 22: Example with Micro Payment Mechanism, Continuity Index



At this point, it would be interesting to introduce an incentive, for example ban the nodes from downloading unless they have a positive score, that means unless they upload as much as they download. The resulting continuity index figure for this scenario can be seen on the left. It is necessary to remind the reader that now the evolution of each node line represents two things on the same time: the mean of the continuity indexes at that particular iteration after 10 simulations and the probability that the play out starts at that iteration. We have already explained the reasons for this. In case of lack of clarity, we also add a table with the mean play out start iteration for each node.

Node	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Mean play out iter.	14.75	6.8	6.7	1.8	-	4.9	3.7	3.1	1	1	1.8	3.4	2.6	2.1	1.5

Table 16: Case 6 with Micro Payment Mechanism, Mean play out start iteration for the 15 nodes

Node 4, starting with an empty buffer, is not able to obtain enough packets to share with others in order to augment his score, so he never succeeds in acquiring the 21 packets necessary to start the play out. Almost the same happens to node 0, whose value 14.75 represents the mean start out iteration for the 4 out of 10 simulations, where he actually gets to start the reproduction of the live stream. In the rest of them (6 out 10) he is starved like node 4. So, in the calculation of the mean play out start iteration for the 10 simulations, these two nodes are left out, and for the rest of them, we get a value of **3.1077.** We observe that there is a considerable amelioration, which is logical since for the majority of the simulations, two nodes are left out so the resources are shared among fewer users. We can also see that this enhancement is impressive for the rich and fast nodes, which are naturally favored by this incentive mechanism as they have spare capacity to bust their score. However, before the experiment, we would expect fewer nodes to be able to reach the point of actually obtaining 21 packets to start the play out, because the policy we use is very restraining. For example, nodes 0-3 should be able to download only 1 piece per iteration and that only if their lucky, so what's the trick behind this unexpectedly good performance? The reason is the existence of the random mechanism, who allows all the nodes to obtain at least 3 packets per iteration, even if they have a negative score or even if they get nothing by their fellow nodes. In the next simulations, the random mechanism is active only if the node has a positive score in order to avoid unexpected results like the previous one.

4.2.3 Study of the Initial Credit Factor

Keeping all the existing parameters in mind and adding this new notion of the random mechanism, the goal of the next section is to examine what happens if we vary the initial score, with which every node joins the system. We can see how the overall performance is affected by its value in the next table.

Initial credit	N=number of nodes who obtain 21 packets in more than 50% of the simulations launched	Mean play out start iteration for the N nodes	Mean continuity index for the N nodes
0	12/15	2.6306	0.8733
10	14/15	3.0143	0.9466
25	15/15	3.58	0.9468
40	15/15	3.69	0.9775
55	15/15	3.58	1
70	15/15	3.5133	1

Table 17: Performance of the system with various initial scores

The first odd element in this table is the fact that the first two cases seem to present a better performance than the rest in terms of mean play out start iteration although their nodes begin with a lower initial credit. This can be easily explained as we must consider that both the mean play out iteration and the continuity index are calculated for the nodes who actually reach the point where the live streaming begins, which means nodes who obtain the first 21 packets before the end of the 20^{th} iteration. In the first two cases one to three nodes are excluded, so naturally the performance of the rest ameliorates. What's more, the excluded nodes are the poorest ones, who

have the worst performance even when they succeed at starting the live streaming, and by consequence their contribution to the mean value is negative.
We can also observe that the mean play out start iteration remains almost the same as the initial credit rises and its raise is reflected on the continuity index, which augments until it reaches a value of 1 for all the nodes for an initial credit of 55, that is for a score which allows the nodes to fill half their buffers without contributing anything.
In order to understand better the evolution of the system, let's look closely at the next graphs for the case where the nodes start with an initial credit of 10 points. In the left part of figure 23, we can see that 5 out of 15 nodes reach a peak and afterwards their continuity indexes start to decrease. We should attribute this to the fact that these nodes use all their credit and then, not having a high enough upload capacity, are unable to raise it in order to reach a positive value, so they are starved. What's more, since they are generally slow nodes, they fill their buffers slower than the others, so even if they wanted to contribute, after a while there is really nobody who is interested in what they have to offer. Maybe they would have a chance to augment their score if there where always new nodes joining the system, who would be interested in the pieces they own.
These assumptions are verified by the right part of figure 23, where we can see the evolution of the scores of the nodes, as the simulation advances. We can observe than only 7 out of 15 nodes converge to a positive value, whereas the rest stay below zero, unable to upload or download after a while. The reason for which this is not reflected to the continuity indexes of all of the 8 nodes with negative scores and affects only 5 of them, is the following: The remaining three are either medium nodes, whose download capacity can guarantee them enough pieces until the 8th iteration where their score becomes and stays constantly negative to last until the 20th iteration without loss of video quality, or their buffers are already almost full so the negative score will affect them later. In my opinion, it is rational to expect that even these three nodes, who seem safe at the moment, will suffer a deterioration in their continuity index in the future.

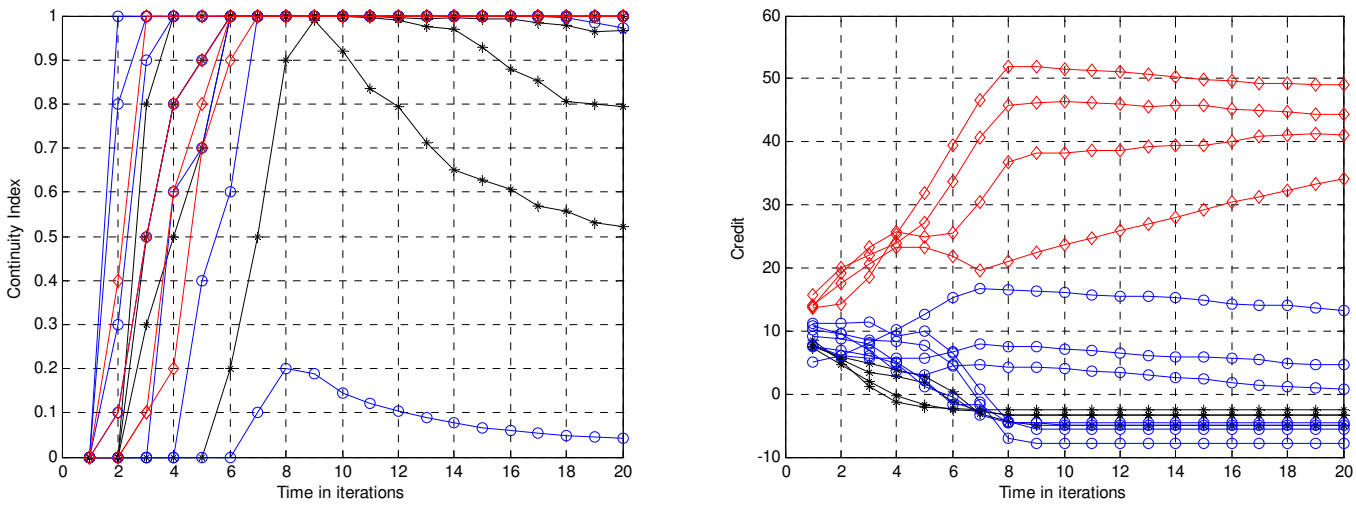


Figure 23: Initial Credit Factor in Micro Payment Mechanism, Initial Credit 10

Now, let's examine the case where all the nodes start with an initial credit of 55. This time, we can see that all the nodes continuity indexes converge to a value of 1 sooner or later. This is verified by figure 24, where it is obvious that 12 out of 15 nodes converge to a value well over zero. There are, though, three nodes who end with negative values. We can expect these nodes either to suffer a performance

deterioration in the future, either their score will fluctuate around zero in order to maintain a stable video quality.

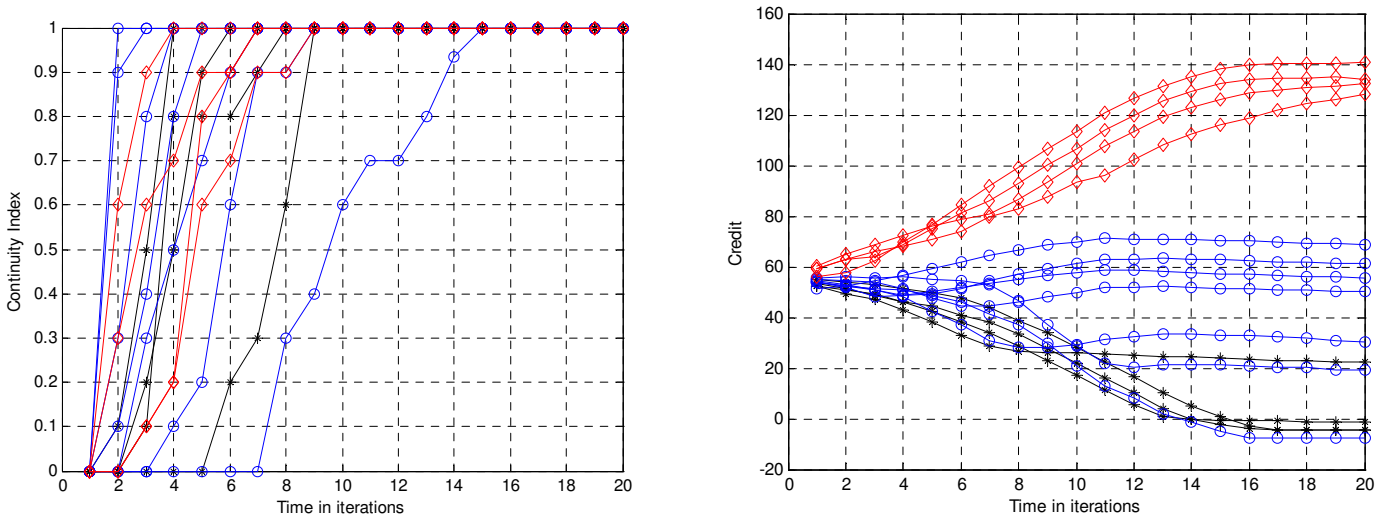


Figure 24: Initial Credit Factor in Micro Payment Mechanism, Initial Credit 55

In conclusion, we can say that even with a quite big initial credit, not all nodes are able to function well, and some are left behind unable to react ant interact with others. So, this micro payment mechanism is rather rigid and doesn't have the necessary flexibility to allow poor or slow nodes to join the system safely. Some ideas as to how we could refine the mechanism with the necessary flexibility and adaptability follow:

The major flaw of this mechanism is the fact that it considers all the peers as equals, which is far but true. The problem is that, whereas all the peers have the same need, a regular media flow with good quality, they don't have the same capacities. A node with a slow dsl connection shouldn't be expected to contribute the same as a peer with a fast campus connection, because simply he can't. A first idea aiming to fix this injustice is to change the way we consider the contribution of each peer, in terms of the data he uploads. Instead of just counting the megabytes of uploaded data, we should rather use as a measure the following percentage: $\frac{data\_uploaded}{upload\_capacity}$. This way, we should have a qualitative measure of the willingness of contribution of a peer and if this percentage is high enough, we should grant the peer access to the stream, even if he may not be in position to upload the same quantities as some of his peers. A practical problem of this proposition is that, in the absence of a central entity, there is no authority that could be responsible to get reliable information concerning the actual upload capacity of each peer. However, we could adopt a solution that is often proposed, even though more complicated, the formation of local neighborhoods, where the upload capacities of each node are calculated by its peers and verified inside the neighborhood with consecutive comparisons.

Another idea is to use the same approach we adopted in the previous differentiated services example. In short, we should compare the available global upload capacity of the system with the fraction that is actually used by the peers, and if there is spare capacity, the poor nodes should be allowed to use it in order to get the stream. This approach is less refined and the quality measure used is less representative of the peers' behavior but on the same time it is more simple to implement and less costly to keep it running.

Generally, a micro payment mechanism with a more flexible and intelligent handling of the peers credits could be promising.

5. Conclusion

During this research internship, I worked on incentive mechanisms for P2P live streaming applications. After the examination of the existing P2P systems and architectures, a series of simulations were done in order to evaluate and compare the existing algorithms and finally propose a differentiated services paradigm.

5.1 Summary of the Work

First, we studied some of the existing algorithms to see if they could be applied in live streaming. Capacity based selection of the next uploader presented better performance than bitmap resemblance based selection, but the best results were observed when we implemented the BitTorrent choke algorithm. However, as we have already explained, this mechanism has a very weak defence against free riders and it's also inappropriate for live streaming because it was build for an environment with multiple seeds and low delay sensitivity. On the contrary, in live streaming there is only one source, which is incapable of serving everybody, and considerable delays before or during the streaming are unacceptable. After testing various scenarios with these algorithms so as to verify their behaviour, we examined a differentiated services example based on a reputation system, which we considered more appropriate for live streaming. This mechanism presented similar results as BitTorrent in the case of fast nodes and slightly worse in the case of slower nodes. It also penalised more severely nodes with low contribution, new comers and free riders. At this point, we were confronted with a dilemma: should we prefer an anti-free rider mechanism, which would also cause problems to new comers, or a more flexible one, which would be easily exploited by malicious users? A definite answer to this question demands profound research and experimentation in real internet conditions, so it exceeds the goals of this internship and rests to be addressed by future work. However, an answer we can give based on the work done during this study, is the following: Live streaming applications are much more resources demanding and delay sensitive than file sharing systems, so the regular contribution and cooperation of all the users are vital to keep the system running. As a result, we would tend to prefer a stricter mechanism that guarantees contribution, than a flexible one which would integrate new comers more easily. Finally, we examined a micro payment example, which presented limited capacity to support a live streaming system, but with the introduction of a more refined micro payment scheme, it could be promising. The conception and implementation of such an algorithm is a good idea for a future application.

5.2 Open Problems, Future Work and Applications

The open problems of this study fall mainly into two categories and they determine also the future work, that could be done in this domain to open new directions.

The first one concerns the refinement of the proposed algorithm or the conception of a new technique which would allow us to distinguish with success free riders from new comers. This is a major problem of P2P systems, maybe it doesn't seem very urgent now because there is enough spare capacity at the end users and the existing applications aren't extremely demanding, but we can suppose that soon, with more and more people having access to the internet and with applications growing more and more "hungry" of resources, free riders exploiting the P2P systems will severely harm their performance. This problem becomes more complicated because of the intelligent nature of free riders, who tend to adapt their habits to fit those of new comers, in order to pass unnoticed by the potential defence mechanisms of P2P

networks. In my opinion, a serious solution to this phenomenon would implicate a kind of learning function, maybe involving a neuron network, in order to create an always evolving profile of a free rider, which would eventually allow us to catch him on the act.
Another direction of this aspect would be the implementation and testing of new fairness mechanisms, which would aim at reinforcing the sentiment of community inside a P2P network. On the contrary, the introduction of more and more strict rules, efficient as it may be, has as a consequence the disintegration of every member of this virtual community, and on the same time malicious users tend to always find a way to trick the system.
Finally, we could focus on the limitations of the mechanisms studied, which were mentioned in the previous paragraph, and try to ameliorate their performance.

The second direction concerns the technique of the simulation itself. I think that the next step of tests would be to incorporate the implemented algorithms in an existing protocol and then repeat the experiments in a real environment, the actual internet. The difficulty encountered at this point is that, in order to derive comprehensive results, a great number of peers should implement the new mechanisms and, what's more, we should be able to watch these peers closely. The only existing environment with such properties is Planet Lab but again, since it is an experimental network with many applications running on the same time, we couldn't really be sure that the results are representative. An intermediate solution would be to repeat the simulations with NS2 or with an existing event driven simulator of a higher level.

Until now few studies have been done on the domain of incentive mechanisms for P2P live streaming, and the research of a definite solution is far from finished. I think that the need for a comparative study of the existing algorithms is urgent, in order to discern the direction of the future research and to allow and promote a creative evolution of today’s systems.

6. Bibliography

[1] “Clustering and Sharing Incentives in BitTorrent Systems”, Arnaud Legout, Nikitas Liogkas, Eddie Kohler, Lixia Zhang
[2] “BiToS: Enhancing BitTorrent for Supporting Streaming Applications”, Aggelos Vlavianos, Marios Iliofotou, Michalis Faloutsos
[3] “Coolstreaming/DONet: A Data-driven Overlay Network for Peer-to-Peer Live Media Streaming”, Xinyan Zhang, Jiangchuan Liu, Bo Li, Tak-Shing Peter Yum
[4] “On Large Scale Peer-to-Peer Live Video Distribution: Coolstreaming and its Preliminary Experimental Results”, Xinyan Zhang, Jiangchuan Liu, Bo Li
[5] “Refine DONet's Overlay with Network Distance Estimation”, Bin Chang, Yuanchun Shi, Nan Zhang
[6] “Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh”, Dejan Kostic, Adolfo Rodriguez, Jeannie Albrecht, Amin Vahdat
[7] “Maintaining High Bandwidth under Dynamic Network Conditions”, Dejan Kostic, Ryan Braud, Charles Killian, Erik Vandekieft, James W. Anderson, Alex Snoeren, Amin Vahdat
[8] “P2P Live Media Streaming: Delivering Data Streams to Massive Audiences within Strict Timing Constraints”, Rapport de Thèse Professionnelle par Fabio Pianese
[9] “The Pulse System: A new P2P prototype for live streaming”, Thesis de Diego Perino
[10] “PULSE: A Novel Unstructured Approach to P2P Live Media Streaming”, Fabio Pianese, E-Next WG3 CDN Workshop
[11] “The PULSE performances under real network conditions”, Diego Perino, Fabio Pianese, MARDI workshop “Models and Algorithms for Decentralized Networks over Internet”
[12] “PULSE, a Flexible P2P Live Streaming System”, Fabio Pianese, Joaquin Keller, Ernst W. Biersack
[13] “Robust Incentive Techniques for Peer-to-Peer Networks”, Michal Feldman, Kevin Lai, Ion Stoica, John Chuang
[14] “Incentive Mechanism for Peer-to-Peer Media Streaming”, Ahsan Habib, John Chuang
[15] ”Incentives for Sharing in Peer-to-Peer Networks”, Philippe Golle, Kevin Leyton-Brown
[16] “Optimizing the Throughput of Data-Driven Peer-to-Peer Streaming”, Meng Zhang, Qian Zhang
[17] “QoS-aware Streaming in Overlay Multicast Considering the Selfishness in Construction Action”, Dan Li, Jianping Wu, Yong Cui, Jiangchuan Liu
[18] “Using Layered Video to Provide Incentives in P2P Live Streaming”, Zhengye Liu, Yanming Shen, Shivendra Panwar, Keith Ross, Yao Wang
[19] “P2P IPTV Measurement: A Comparison Study”, Thomas Silverston, Olivier Fourmaux
[20] “Source vs Data-Driven Approach for Live P2P Streaming”, Thomas Silverston, Olivier Fourmaux
[21] “Community Building over Neighborhood Wireless Mesh Networs”, Panayotis Antoniadis, Johanna Chouffane, Benedicte Le Grand, Anna Satsiou, Leandros Tassiulas, Rui Aguiar, Jao Paulo Barraca, Susana Sargento

7. Appendix

7.1 Main Program Pseudo Code

```
Global Variables

Vecarray-> Each line represents the buffer of a node
Globalbuffer->Each line represents the bitmap of a node, that is the interest window
Upload->Each element represents the upload limit of each node, periodically updated
```

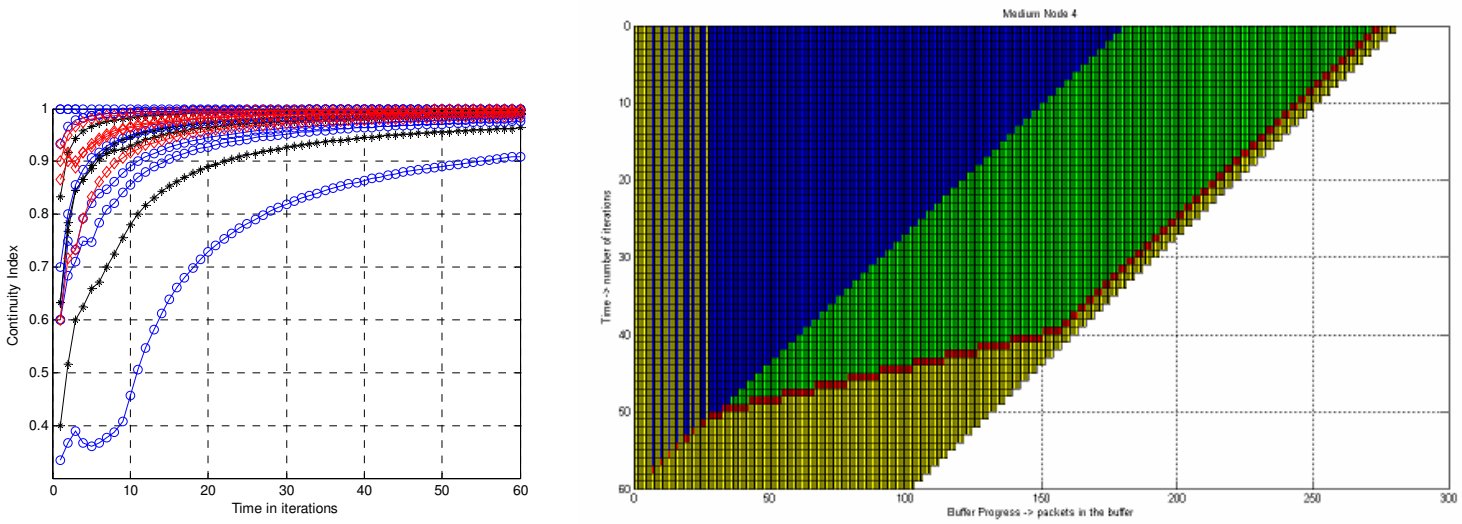
```
Node pseudo

Wait until all threads have started
While inbound>0 do
      If all nodes have been used, exit
      If the buffer is full, exit
      If there are no available resources, exit
      Choose a not already used node at random beginning with those with those with the biggest upload rate
      Load its bitmap
      Find a missing block available in that node's bitmap
      Demand it
      Wait RTT to get the answer
      If its upload limit hasn't been reached
            Update your bitmap
            Decrease other node's upload limit
            Decrease your own download limit
            Continue with other missing blocks
            If you have already used one third of your download     bandwidth on one node, change node
      Endif
Endwhile
If your download limit hasn't been reached yet, do for as many pieces as you have already downloaded+3
      Use a certain distribution to obtain packets from your neighbors outside the 15 known nodes
      Update your bitmap
      Decrease your download limit
Endif
Wait until all thread have finished
Update bugger vector with the new bitmap
Slide the interest window
Update the globalbuffer with new bitmap
```

```
Main pseudo

Create the RTT array
Create random bitmaps for the nodes
Attribute random number of packets to each node in random positions in the bitmap
While there is change in the globalbuffer
      Create and start threads with proper parameters
      Guarantee that all threads start and finish simultaneously
      Wait for every thread to finish
      Continue with updated bitmaps at each new iteration
Endwhile
```

7.2 Upload Capacity Based Selection: Figures for 60 Iterations



Left Continuity Index, Right Buffer Progress of node 4