Partial Video Replication for Peer-to-peer Streaming

Sailaja Uppalapati and Ali Şaman Tosun

Department of Computer Science
University of Texas at San Antonio
San Antonio, TX 78249
{suppalap,tosun}@cs.utsa.edu

Abstract. Video streaming over peer-to-peer networks has attracted a lot of interest recently. However most of the research on streaming in peer-to-peer networks focused on schemes where all the clients have the whole movie. In this paper we propose schemes where clients store only partial movie after viewing the movie. We propose cooperative schemes where replication is done in a way that maximizes a global function and uncooperative schemes where each node makes replication decision independently. We evaluate both schemes using extensive simulation. Simulation results show that cooperative schemes perform better but they are harder to implement and maintain. Uncooperative schemes are simpler, based on a distributed algorithm but they suffer from lower performance.

1 Introduction

Peer-to-peer (P2P) is a new paradigm in which each peer stores the movie after streaming and act as a supplying peer by streaming the movie to other requesting peers thus serving both as a client and as a server. Combined storage of large number of peers allows users to locate a wide variety of multimedia content on the P2P network. The popularity of P2P networks and high number of peers on P2P networks with high-speed Internet connections have fueled interest to stream video over P2P networks. There are many challenges introduced when streaming is done using P2P paradigm as opposed to client-server paradigm which suffers from single point of failure, performance bottlenecks as media is centralized at the server.

Recently, streaming media from multiple sources has received a lot of attention. When all the nodes store the whole video, packets can be retrieved from the node which minimizes loss and delay [6]. The probability of packet loss in bursty environment is reduced by using FEC [7] where the source sends multiple redundant packets to the receiver. The receiver can reconstruct the original packets upon receiving a fraction of the total packets. PeerCast[3] streams live media using an overlay tree formed by clients and CoopNet [8] proposes a mechanisms for cooperation of clients to distribute streaming video when server is overloaded. A peer-to-peer media streaming model with an optimal media data assignment algorithm and a differentiated admission control protocol is proposed [14] assuming that all the peers store the whole video. A hybrid architecture that integrates Content Distribution Network(CDN) and P2P based media distribution given in [13]. CDN has a number of CDN servers deployed and the client can request media from the closest CDN server. Layered peer-to-peer streaming is proposed to handle asynchrony of user requests and heterogeneity of peer network bandwidth [2]. Administrative organization of peers to reduce control overhead in media streaming is proposed in [12]. Many P2P networks like CAN [9], CHORD [11] and Pastry [10] were proposed to perform peer lookups. Promise peer-to-peer system [4] supports peer lookup, peer-based aggregated streaming and dynamic adaptations to network and peer conditions. Gnustream [5] is a receiver-driven media streaming system built on top of Gnutella. Splitstream [1] distributes the forwarding load among all the nodes and accommodates peers with different bandwidth capacities by constructing a forest of interior-node-disjoint multicast trees.

All of the above techniques assumes that the whole movie is stored at all the peers and focussed on how to choose the peers based on their delay, loss and outgoing bandwidth. They didnot consider the case where peers have limited storage and may not be able to store the entire movie. If peers store partial video, a whole new set of challenges are introduced including the following

– How can a client determine whether a given set of peers are enough to stream the video?
– Given space for k segments, how can a peer determine the k segments that it stores?
– Should a peer cooperate with other peers to determine which segments it stores?
– How much control information needs to be exchanged to determine the supplying peers when a client requests a movie?

In this paper, we investigate the above issues. We propose two classes of schemes: cooperative and uncooperative. In cooperative schemes peers exchange information with each other and segments that are to be replicated are the ones that maximize the global utility function. In uncooperative schemes no information is exchanged between peers and each peer independently makes a decision on which segments it stores. We evaluate both schemes using extensive simulation. Cooperative schemes are complex to implement. However, they perform much better. Uncooperative schemes requires no coordination and are simpler to implement. This comes at the cost of lower streaming sessions that can be supported simultaneously. We also propose region-based cooperative scheme to reduce the overhead of cooperative schemes and to make them more scalable.

The rest of the paper is organized as follows: In section 2 we describe the cooperative and uncooperative schemes. We provide experimental results in section 3 and discuss pros and cons of each in section 4. Finally, we conclude with section 5.

2 Proposed Schemes

We assume the following to simplify the problem. Each movie consists of M segments labeled S_1 to S_M and each segment takes a weight between [0..1]. In the homogeneous case the weights of all the segments are equal. To simplify the problem we consider the homogeneous case. Each peer stores partial movie after streaming the whole movie. Partial storage is based on segments and each peer stores a subset of the segments. Peers are denoted by P_i and fraction of video stored at peer P_i is denoted by f_i. Each peer determines the value of f_i based on available disk space and outgoing bandwidth. If the consumption rate of the movie is B Mbps and outgoing bandwidth is $\frac{B}{4}$ Mbps then f_i should be $\leq \frac{1}{4}$. On the other hand, f_i should be set in such a way that storing f_i fraction of movie does not exceed the available disk space at the peer.

We assume the following in proposed schemes: peers can join and leave the peer-to-peer network, peers have limited storage and may not be able to store the whole video.

2.1 Uncooperative Schemes

Uncooperative schemes involves no communication between nodes to determine which segments to replicate. Since nodes may join and leave the peer-to-peer network at any time uncooperative schemes are interesting. In addition, uncooperative schemes are simpler.

In uncooperative schemes the peer P_i first determines the value of f_i and the picks a random number seed s_i. Let $rand(s_i, n)$ denote the n^{th} random number generated by seed s_i. We assume that random numbers generated are in the range [0..1]. Peer P_i stores segment j if $rand(s_i, j) \ <= \ f_i$. As a result each segment is stored with probability f_i. So, expected number of segments stored is $f_i M$ where M is the number of segments.

When a peer sends a streaming request, each peer P_i responds with the pair (f_i, s_i). The peer needs to make a decision based on the pairs received. The peer chooses a subset of peers to stream the video. Ideally, the subset of peers selected should satisfy two constraints. Every segment should be stored by at least one peer and the set should be as small as possible. However, since storage decision is made probabilistically, it is not possible to guarantee that every segment be stored by at least one peer. Assume k peers given by $\{P_1, ..., P_k\}$. First fragment is not stored in these peers with probability $P_e = \prod_{i=1}^{i=k}(1 - f_i)$. So, no matter how many peers we have, there is a small probability that a fragment may not be available in them. In such cases, we request the missing fragment from the original source. Since source needs to handle too many requests, we want to limit the probability of contacting the source. We have a threshold and select peers in such a way that the expected number of segments that need to be retrieved from the source is less than the threshold.

Peer selection problem can be stated formally as follows:

Peer Selection Problem: *Consider a movie with M segments. Given a set of peers* $\{P_1, P_2, ..., P_N\}$ *with each peer having* (s_i, f_i)*. Find the smallest subset* $\{P_1, ..., P_k\}$ *of peers such that* $M \prod_{i=1}^{i=k}(1 - f_i) \leq threshold$*.*

Peer selection problem can be solved efficiently by sorting the f_i's in decreasing order and by choosing the largest values until $M \prod_{i=1}^{i=k}(1 - f_i) \leq threshold.$

In uncooperative schemes, requesting client broadcasts a message to the peer-to-peer network (limited broadcast with increasing ranges) and receives the pair $((s_i, f_i))$ from each corresponding peer. It solves the peer selection problem and determines a set of peers. By using the information embedded in $((s_i, f_i))$ it can determine which peers store which segments. Peer i stores segment j if $rand(si, j) \ < \ f_i$ where $rand(a, b)$ is the b^{th} random number generated with seed a. Control messages in uncooperative scheme are quite short and it is possible to use aggregation techniques to reduce the number of messages.

2.2 Cooperative Schemes

In cooperative schemes, the segments that are stored after streaming the video are chosen based on a global optimization criteria using the utility function. The segments that maximize the utility functions are chosen for storage.

We next discuss the desirable properties of utility function and then propose some functions that meet the properties. Users view the movie from the beginning till the end and it is better to replicate segments that are stored at fewer nodes. As the number of copies of a segment increases, the potential gain from one more copy decreases. Having 4 copies of a segment instead of 3 copies is great. However, there is no big difference in having 200 copies of a segment versus 199 copies of a segment in the network. Utility function is also independent for each movie since a client who streams and views a video can only store that video.

Utility function for movie m is denoted by U_m and utility of a movie is the sum of utilities of all the segments in the network. Utility of segment S_i is denoted by v_i. Therefore, utility function of movie m is given by

$$U_m = \sum_{k=1}^{M} v_k \tag{1}$$

As the number of copies of a segment increases, the utility value should increase but at a much slower rate. We use the following function for v_k.

$$v_k = \sum_{i=1}^{c_k} \frac{1}{i} \tag{2}$$

The *segment vector* $C_m = (c_1, c_2, ..., c_M)$ denotes the number of copies of each segment available in the network for movie m. c_i denotes the number of copies of i^{th} segment available in the network. So, utility value for movie m is

$$U_m = \sum_{k=1}^{M} \sum_{i=1}^{c_a} \frac{1}{i} \tag{3}$$

In cooperative schemes, original source of the movie maintains the segment vector and each peer determines the segments to replicate according to the utility function. The segments that maximize the utility value are chosen for storage. Increasing the number of copies of segment i from c_i to c_{i+1} increases the utility value by $\frac{1}{c_i+1}$. Therefore, to maximize the utility value the segment that needs to be replicated should have the smallest value of c_i and the segment to replicate is determined by the equation $x = argmin_k \frac{1}{c_k}$. If multiple segments are to be replicated then the equation is solved again with updated segment vector C_m.

There are many challenges in implementation of cooperative schemes. The vector C_m needs to be computed and this computation requires input from all the peers who has the movie. When a peer leaves the network the vector C_m needs to be updated. When a node fails the vector C_m will not be accurate. To handle these problems proposed scheme assigns a leader to each movie. This leader maintains C_m and handles update and retrieval requests for C_m. When a node decides to leave the network, it sends a message to the leader. When a node starts a streaming session, it sends a message to the leader indicating its value of f_i and requesting the vector C_m. The leader can compute the segments that the peer will choose for replication by solving the replication equation. The leader updates the vector C_m accordingly.

Each node stores an *local vector* D_m whose entries are 0-1s and indicate whether segment i of movie m is stored by the peer or not. The leader stores the local vectors of all the clients in the system who partially replicate the movie. When a client requests the movie, the leader uses this information to find a subset of peers who can stream the video to the client. The subset of peers has to satisfy two conditions. First, each segment should be stored by at least one peer. Second, the set of peers should be as small as possible to reduce control message overhead and to improve utilization of our system. In proposed scheme, a peer can supply video to a single node at a given time (since f_i determined according to outgoing bandwidth). Therefore, minimizing set of supplying peers is required. Peer selection problem can be stated formally as follows:

Peer Selection Problem: *Given a set of peers* $\{P_1, P_2, ..., P_N\}$ *with each peer having local vector* D_m*. Find the smallest subset of peers such that each segment is available by at least one peer.*

Peer selection problem can be reduced to *set cover* problem and is NP-complete. Think of the segments stored at a peer as a subset and the problem is to find the smallest set of subsets that cover the whole set. We used the greedy heuristic given in figure 1 to solve the peer selection problem. In this algorithm X denotes the set of segments $\{1, 2, ..., M\}$ and F denotes a family of sets where the segments stored by each peer is an element of this family. Since this heuristic requires input from all the nodes, the leader solves the peer selection problem. When a node requests a movie, it sends a message to the leader. If the set of active nodes in the system can grant this request, the client and the supplying peers are contacted to inform what they need to do. Supplying peers are then placed on the inactive list for the duration of the streaming session.

```
Greedy-Set-Cover(X,F)
01 U ← X
02 C ← ∅
03 while U ≠ ∅
04         select an S ∈ F that maximizes | S ∩ U |
05         U ← U-S
06         C ← C ∪ {S}
07 return C
```

Fig. 1. Greedy Set Cover Algorithm

The leader stores the local vector received from each client. In addition the leader maintains a status list. So, the amount of space required at the leader is $O(AM)$ where A is the number of nodes currently connected to the peer-to-peer network and partially store the movie, M is the number of segments in the movie. Storage requirement is reasonable and a typical machine can handle hundreds of thousands of clients.

Greedy heuristic of choosing the peer that has the maximum number of unselected segments has an approximation ratio of $H_P = \sum_{i=1}^{P} \frac{1}{i}$ where P is the largest set in F. This means that number of peers selected by greedy algorithm can be at most H_P times more than the number of peers in optimal solution. In our case the number of segments determines the approximation ratio. With M segments the approximation ratio is $H(M)$.

Since greedy algorithm will be executed whenever a client requests a movie, efficient implementation is crucial. It is possible to implement greedy set cover to run in $O(\sum_{S \in F} \mid S \mid)$ time. So, running time is proportional to the total number of segments stored in the network.

If the leader is able to find the set cover for a request, it sends the list of peers in the set cover and their local vectors to the requesting client. Requesting client contacts the peers and streams the video from them. It is possible that a segment is stored at multiple peers in the set cover, in this case the requesting client can pick a peer based on some other criteria such as delay or number of hops.

2.3 Region Based Cooperative Scheme

When the network grows a single leader may not be able to handle all the clients requests. To minimize the load on the leader we divide the network into regions with each region having a regional leader.

Splitting of a region means moving some of the nodes of region R to region R$'$. This is done only when both the regions R and R$'$ can handle client requests independently otherwise we ignore the idea of splitting. To make the splitting decision we consider the distance between peers and the network conditions. The distance between peers is determined by the number of common segments that exists between the two peers. Since the number of common segments between peers may be large we normalize it by taking value between $a = 0$ and 1. The network conditions can be the delay and bandwidth of the network in consideration and let this value be between $b = 0$ and 1. Thus the cost of edge between peers is $a\alpha + b\beta$ where $\alpha + \beta = 1$, α and β are user defined parameters.

For splitting a region the following challenges are to be addressed.

- When the splitting of a region can be done?
- How to pick a leader for region R$'$?
- How to create balanced self-sufficient regions?

The splitting algorithm shown in figure 2 takes a region R and the parameters a and b. The Split-Region algorithm can be called if min count is greater than M$'$ otherwise the region is too small to split. Here M$'$, D$'$ are user defined parameters and min count is the minimum number of copies of fragment i in system. The line 8 selects the leader of region R$'$ by choosing one node at random from the candidate set and line 9 moves the node to region R$'$ from region R making it the leader of R$'$ as it is closer to R$'$. The algorithm continues to move nodes from region R to region R$'$ until the set cover for R$'$ can be done or the number of nodes currently in R$'$ is less than low-threshold. If the number of nodes in region R$'$ is greater than high-threshold, a balanced split is not obtained so we abort the split test and merge the nodes of region R and R$'$. The low-threshold and high-threshold are used to obtain balanced self sufficient regions.

```
Split-Region(R,a,b)
01 p ← leader(R)
02 for each i ∈ R
03     $d_{i,p}$ = Compute-Distance(i,p)
04 C ← ∅
05 for each i ∈ R
06     if $d_{i,p} > D'$
07         C ← C ∪ {node i}
08 q ← Select-Leader(C)
09 R′ ← {q}
10 for each i ∈ R
11     $d_{i,q}$ = Compute-Distance(i,q)
12     if $d_{i,p} > d_{i,q}$
13         R′ ← R′ ∪ {node i}
14         R ← R − {node i}
15 if SET-COVER(R′) = False
16     for each i ∈ R
17         $d_{i,q}$ = Compute-Distance(i,q)
18     D = Sort $d_{i,q}$ in increasing order
19     for each i ∈ D
20         if SET-COVER(R′) = False or num-of-nodes(R′) < low-threshold
21             R′ ← R′ ∪ {node i}
22             R ← R − {node i}
23         if num-of-nodes(R′) > high-threshold or SET-COVER(R) = False
24             Abort Split test
21 return
```

Fig. 2. Split Region Algorithm

3 Experimental Results

We simulated both cooperative and uncooperative schemes using extensive simulation. Simulation is written using csim. Initially only the source node has the movie and movie requests arrive according to poisson distribution. The simulation is done with three values of $\lambda = 0.001, 0.0001$ and 0.0003. Movie is divided into 100 segments and clients store some of the segments after streaming. The results were based on homogeneous case where all the segments are of equal weight, heterogeneous case is part of future work. 50% of the clients store 10-20 segments and the remaining 50% store 20-50 segments. 20% of the clients who join the network remain in the network till the end of simulation. Since the peers can set their fraction of video stored based on outgoing bandwidth, peers can supply video to at most one client at a given time. Since storing segments involve control overhead in addition to storage overhead, it is not feasible to store a few segments. In simulation, minimum number of segments stored at a node is 10 corresponding to 10% of movie. Similarly, missing a few segments is not feasible since the nodes need to be involved in peer-selection problem. We assume that nodes store at most 50 segments corresponding to 50% of the movie. Movie length is 120 minutes. We have two sets of results. In the first set, after streaming nodes stay in the network till the end of simulation. In the second set, 80% of the nodes stay in the network for 5-24 hours and 20% of nodes stay in the network till the end of the simulation. Simulation length is 1 month.

We use the following metrics to compare the performance of cooperative and uncooperative schemes.

- *min-max count:* Let n_i be the number of copies of segment i in the system. *Min* is given by $\min_{i=1}^{100} n_i$ and *max* is given by $\max_{i=1}^{100} n_i$. Ideally, we want the gap between *min* and *max* to be low.
- *utility value:* Value of utility function. Smooth curves in utility graph are desirable since they correspond to smooth transitions when peers leave the network.
- *number of supplying peers:* Average number of supplying peers involved in streaming sessions. Smaller number of supplying peers are desirable since streaming client needs to manage streaming session with all of them.
- *success-failure count:* Requesting peers resend their request every 5 minutes and leave the network if streaming is not feasible in 20 minutes. *success* denotes the number of streaming sessions successfully completed during the simulation and *failure* denotes the number of nodes that leave the network without streaming.

4 Discussion

The graphs for cooperative and uncooperative schemes are similar when all the nodes stay in the network till the end of simulation. They differ in case of 5-24hrs. So we mainly focus on graphs for 5-24hrs .

Figures 3 and 4 shows the increase in utility as more and more nodes join and the decrease in utility when the nodes leave. This decrease in utility is recovered by new nodes joining the network. In figure 3 uncooperative case between 0 to 5 days there is sudden raise and fall in utility. This is because threshold was initially set to 20 as the number of missing segments will be high. Later when we know that the peers have enough segments we set the threshold to a small value thereby limiting the probability of contacting the source. Min Max values are shown in figure 8 and figure 9. As expected the difference between min and max values are low in case of cooperative whereas in uncooperative, it is large due to the lack of coordination between nodes. Success and failure count shown in figures 5 and 6. Initially the number of streaming sessions is low since only leader has the movie and many clients leave the network without getting the movie. As content is replicated success count increases and failure count decreases. In case of $\lambda = 0.0001$ and $\lambda = 0.0003$, the success and failure is not that significant since the arrival rate i.e. the rate at which the nodes join is low. By the time the new node join the network the existing nodes might leave and the new node is not able to do the streaming with the available nodes thus increasing the failure rate. In case of $\lambda = 0.001$ the success and the failure rate can be seen more significantly.

The supplying peers graphs shown in figure 7 indicates the average number of peers involved in streaming session. Graph shows sudden raises and falls. This is because when a new node joins the network to satisfy its streaming request the peers in the smallest set cover might be busy in streaming session with other nodes so its request is satisfied by the available peers forming a set cover. The peer size graphs shown in figure 10 specifies the average number of peers holding copies of a segment. Based on the network delay, bandwidth client can select the peer with high bandwidth and less delay. Multiple peers having a copy of a segment is 40% in cooperative case whereas in uncooperative it is 80%. This shows uncooperative scheme is better even though its supplying peer size is large. When $\lambda = 0.0001$, this is not true as the arrival rate is low. Most of the incoming clients will be streamed by the leader, since the existing clients may leave the network as their time expires.

In figure 11 the graph on left shows the region size when $\lambda = 0.001$ and graph on right is for $\lambda = 0.0003$. Each line in the graph corresponds to a region. From the graph we see whenever a region size decreases it indicates a split of that region causing a new region to start. In case of $\lambda = 0.001$, at the end of the simulation we obtained 14 regions with minimum region size of 144 nodes and maximum region size of 282 nodes. Similarly for $\lambda = 0.0003$, we obtained 6 regions with minimum region size of 44 nodes and maximum region size of 171 nodes.

Cooperative schemes perform better than uncooperative schemes in terms of number of streaming sessions that can be supported by the system. However, large scale implementation of cooperative schemes is not feasible since nodes selected for set-cover can be far away from each other. Centralized nature of cooperative schemes is also problematic since failure of leader will render the streaming impossible.

Uncooperative schemes perform reasonably and can be implemented in a distributed way using limited range broadcast on the peer-to-peer network. Having a larger set of supplying peers increases the possibility of more than one peer having a copy of a segment. The decision of which peer to choose can be made based on other decisions such as network delay and the number of hops.

5 Conclusion

In this paper, we investigate partial replication strategies for streaming video over peer-to-peer networks. Each client stores partial video after streaming depending on its available disk space and outgoing bandwidth. We propose cooperative schemes where the replication is done in a way to maximize the utility function and uncooperative shemes where the replication is done independent of what is stored at other nodes. Cooperative schemes perform much better than uncooperative schemes. However, cooperative schemes are more complex and requires execution of greedy set cover algorithm for each arriving request. Uncooperative schemes on the other can be implemented in a simple and distributed way. Future work includes investigation of hybrid schemes that combines the benefits of these two by using regional leaders or a hierarchy.

References

1. Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowston, and Atul Singh. Splitstream: High-bandwidth multicast in cooperative environments. In

SOSP'03, October 2003.
2. Yi Cui and Klara Nahrstedt. Layered peer-to-peer streaming. In *NOSSDAV 2003*.
3. H. Deshpande, M Bavea, and H. Garcia-Mollina. Streaming live media over peers. Technical report, Stanford Database Group Technical Report (2002-21).
4. Mohamed Hefeeda, Ahsan Habib, Boyan Botev, Dongyan Xu, and Bharat Bhargava. Promise: Peer-to-peer media streaming using collectcast. In *ACM Multimedia 2003*.
5. Xuxian Jiang, Yu Dong, Dongyan Xu, and Bharat Bhargava. Gnustream: A p2p media streaming system prototype. In *IEEE International Conference on Multimedia and Expo (ICME 2003)*, 2003.
6. T. P. Nguyen and A Zakhor. Distributed video streaming over internet. In *SPIE/ACM MMCN 2002*.
7. Thinh Nguyen and Avideh Zakhor. Distributed video streaming with forward error correction. In *Packetvideo Workshop 2002*.
8. V.N. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai. Distributed streaming media content using cooperative networking. In *NOSSDAV 2002*.
9. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *ACM SIGCOMM*, August 2001.
10. Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, 2001.
11. I. Stoica, R. Morris, D. Karger, F. Kaashoek, and Balakrishnan H. Chord: A scalable peer-to-peer lookup service for internet applications. In *ACM SIGCOMM*, August 2001.
12. Duc Tran, Kien Hua, and Tai Do. A peer-to-peer architecture for media streaming. *Journal in Selected Areas in Communications, Special Issue on Advances in Service Overlay Networks*, 22(1):121–133, January 2004.
13. Dongya Xu, Heung-Keung Chai, Rosenberg Catherine, and Sunil Kulkarni. Analysis of a hybrid architecture for cost-effective streaming media. In *SPIE/ACM Conference on Multimedia Computing and Networking (MMCN 2003)*.
14. Dongyan Xu, Mohamed Hefeeda, Susanne Hambrusch, and Bharat Bhargava. On peer-to-peer media streaming. In *IEEE International Conference on Distributed Computing Systems (ICDCS 2002)*, July 2002.

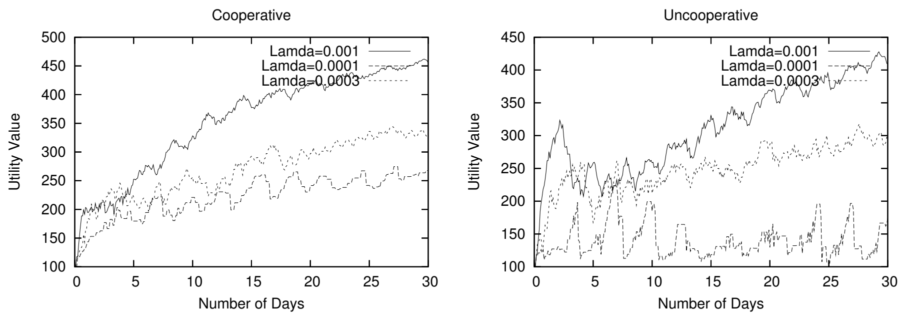

Fig. 3. Utility values (5-24hrs)

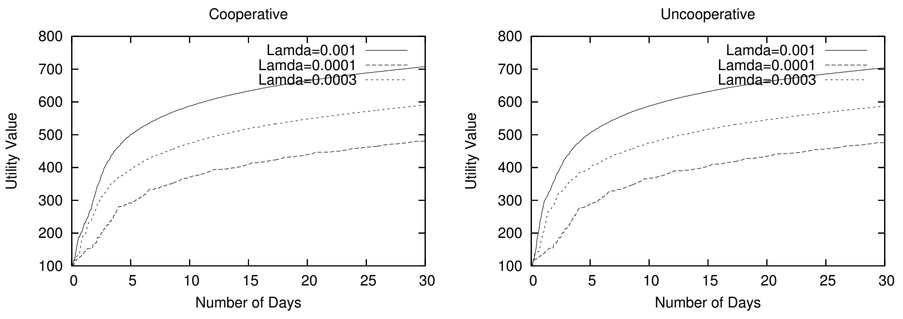

Fig. 4. Utility values (1 month)

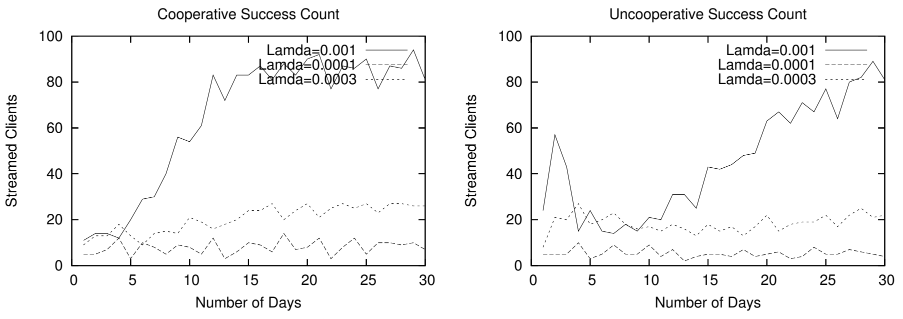

Fig. 5. Success count (5-24hrs)

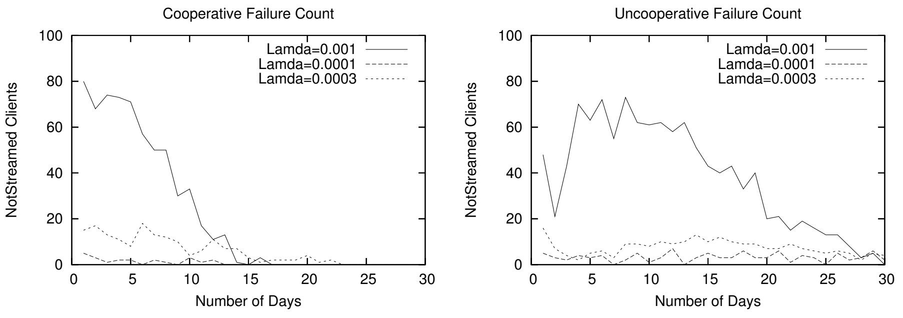

Fig. 6. Failure count (5-24hrs)

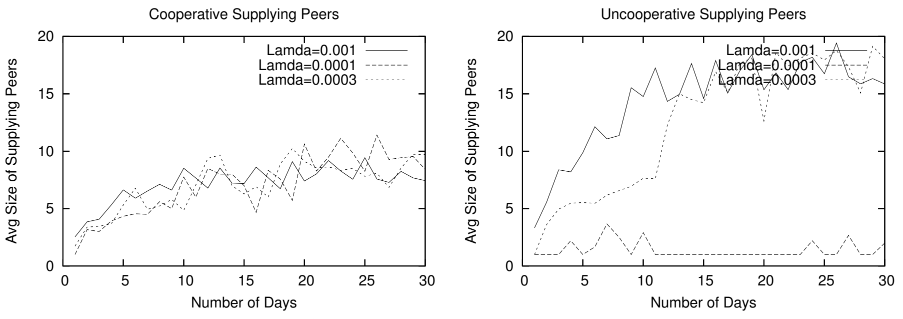

Fig. 7. Supplying peers (5-24hrs)

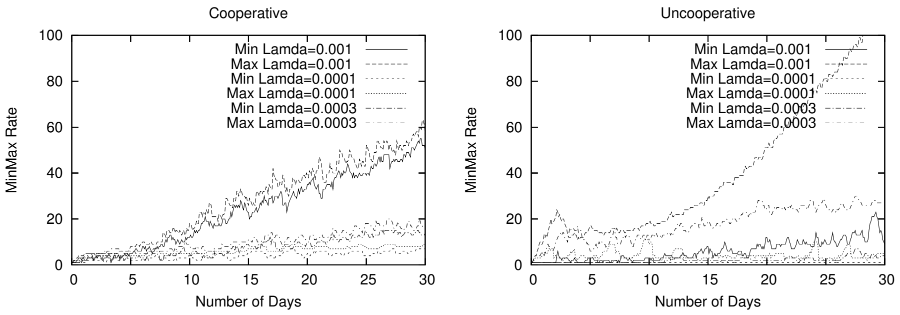

Fig. 8. MinMax values (5-24hrs)

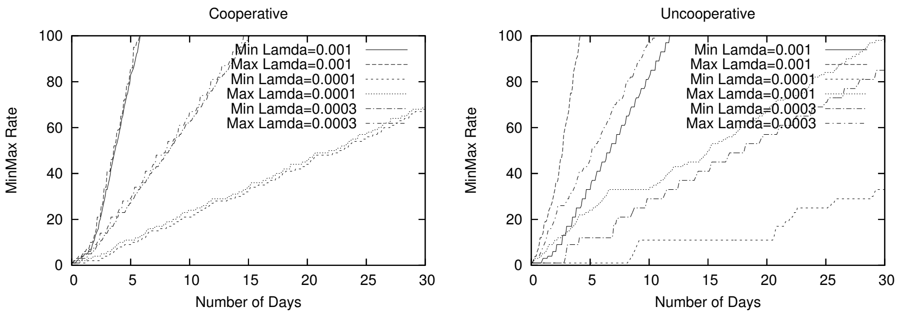

Fig. 9. MinMax values (1 month)

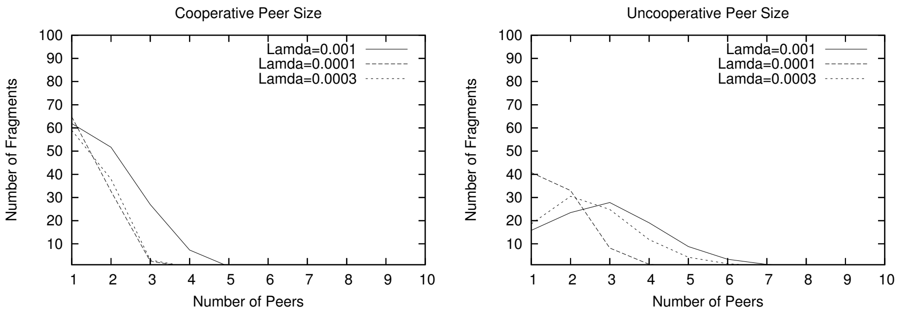

Fig. 10. Peer Size Graphs (5-24hrs)

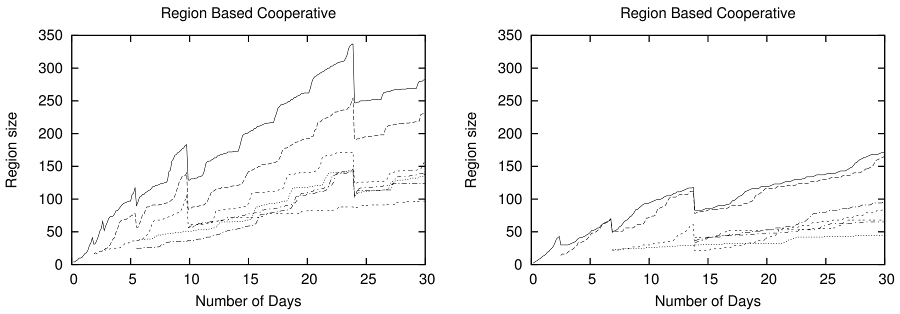

Fig. 11. Region Graphs