

Robust Live Media Streaming in Swarms

Thomas Locher, Remo Meier,
Roger Wattenhofer
Computer Engineering and Networks Laboratory
ETH Zurich
8092 Zurich, Switzerland
{lochert,remmeier,wattenhofer}@tik.ee.ethz.ch

Stefan Schmid
Chair for Efficient Algorithms
TU Munich
85748 Garching
schmiste@in.tum.de

ABSTRACT

Data dissemination in decentralized networks is often realized by using some form of swarming technique. Swarming enables nodes to gather dynamically in order to fulfill a certain task collaboratively and to exchange resources (typically pieces of files or packets of a multimedia data stream). As in most distributed systems, swarming applications face the problem that the nodes in a network have heterogeneous capabilities or act selfishly. We investigate the problem of efficient live data dissemination (e.g., TV streams) in swarms. The live streams should be distributed in such a way that only nodes with sufficiently large contributions to the system are able to fully receive it—even in the presence of freeloading nodes or nodes that upload substantially less than required to sustain the multimedia stream. In contrast, uncooperative nodes cannot properly receive the data stream as they are unable to fill their data buffers in time, incentivizing a fair sharing of resources. If the number of selfish nodes increases, our emulation results reveal that the situation steadily deteriorates for them, while obedient nodes continue to receive virtually all packets in time.

Categories and Subject Descriptors

C.2 [Computer Systems Organization]: Computer-Communication Networks

General Terms

Algorithms, Security, Reliability

1. INTRODUCTION

Over the past years, decentralized systems have continued to replace traditional client-server based systems as the preferred method to quickly and efficiently disseminate bulk data. Likewise, in live media distribution, the same paradigm shift from centralized to decentralized and self-organizing systems can be observed (e.g., several commercial products for Internet television such as *JumpTV*¹ or *PPLive*² use a decentralized architecture).

¹See <http://www.jump.tv/>.

²See <http://www.pplive.com/>.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

NOSSDAV'09, June 3–5, 2009, Williamsburg, Virginia, USA.
Copyright 2009 ACM 978-1-60558-433-1/09/06 ...\$5.00.

A popular technique for data dissemination in decentralized systems is *swarming*: nodes with similar interests dynamically gather in order to fulfill a particular task or to share certain resources. For this purpose, the nodes build an *overlay network* as a virtual routing infrastructure. Swarming has its roots in peer-to-peer systems, but has become a technique of its own merit. The swarming paradigm is expected to play an increasingly important role in the Internet in the future.

Unlike in classic content distribution applications, nodes in a swarm participate in the distribution of data and thereby alleviate the content servers. In case of live data (e.g., live media streaming), small buffers are used to hold the data to be played in the next few seconds, and all data blocks have to be received *in time*. If a given data block is not available locally when the buffer content is to be fetched, an *underflow* will occur, resulting in a reduced playback quality. Hence, the required *liveness* of all data blocks is crucial.

Since any system that depends on the correct functioning of a centralized authority also exhibits a single point of failure, decentralized systems are the preferred solution for various purposes. However, in contrast to a centrally monitored system, a decentralized system has the disadvantage that the notorious problem of *selfish nodes*, i.e., nodes that act in their own best interest, cannot be handled easily. In fact, as decentralized systems depend on the correct behavior of the nodes, ensuring that it is in the best interest of selfish nodes to adhere to the given protocols is one of the most essential problems in decentralized networking. Since there is a trend towards decentralization in many areas of networking, the problem of selfishness will likely gain increasing importance in the future. In the context of efficient data dissemination, a selfish node may consume resources but refuse to offer any service to the system by uploading data to other nodes. While this normally results in an overall loss of efficiency as the average download time for any given file in the system increases, these so-called *free riders* can greatly harm the quality of a broadcast for potentially all nodes in *live streaming* applications due to the underflows they inflict by refraining from providing the required data blocks to their neighboring nodes.

In some sense, eliminating free riders from live streaming systems is easier than in on-demand content distribution systems. If it can be guaranteed, by taking appropriate measures, that free riders are much more likely to suffer from underflows, they will probably leave the system quickly as the experienced playback quality is poor. Moreover, due to the typical repeated interactions between the same nodes, tit-for-tat-based mechanisms can be employed and thus free riders can be identified and penalized. Nevertheless, providing incentives to cooperate in live streaming swarms is still a challenging problem mainly due to the stringent timing constraints.

Thus, the goal of our mechanism for live streaming in swarms is to identify free riders quickly and to guarantee that these nodes receive a restricted amount of data such that it is not worthwhile for

them to remain in the system. It must further ensure that, even if a certain unavoidable amount of bandwidth is wasted, all the obedient and resource-rich nodes do not suffer from underflows, i.e., the quality they experience does not deteriorate in the presence of selfish nodes.

In this paper, we argue that the neighbor selection strategy and the data distribution paths strongly influence the success of any incentive mechanism. In order to illustrate this point, we implemented our strategies and performed extensive emulations. By adequately tuning the incentive mechanism, we achieve a high level of robustness in that obedient nodes always have a satisfactory streaming quality even if a large percentage of all nodes do not provide current data blocks quickly enough or no blocks at all. Moreover, selfish nodes are caught rapidly, and other nodes cease to provide data blocks to them.

The remainder of this paper is organized as follows. In the subsequent section (Section 2), related literature is reviewed. Our fairness mechanism is presented in detail in Section 3; results of our emulations are discussed in Section 4. Finally, the paper concludes in Section 5.

2. RELATED WORK

Free riding is a threat to every system depending on the contributions of all its participants. The existence of free riders in peer-to-peer (p2p) file sharing systems has been pointed out in several papers, e.g., [1, 7]. In the last few years, several mechanisms to deter nodes from freeloading have been proposed, although mainly in the context of peer-to-peer file sharing applications.

One approach to cope with selfish nodes is to maintain private histories of good and bad nodes [5, 9]. The limited scalability of this approach and also the zero-cost nature of online identities in most systems, which can be exploited by creating new identities when needed (white-washing) or by creating several identities at the same time (*Sybil attack* [3]), renders this solution fairly ineffective. The same holds for rank-based node-selection mechanisms [6] where, using scores, cooperation is achieved through service differentiation. A drawback of this approach is that it is difficult and costly to ensure that malicious nodes cannot tamper with the used scores. Moreover, the preference of the best suppliers might quickly result in a clustered topology which might cause the entire swarm to come apart. Thus, it is hard to maintain a robust swarm topology with a bounded delay, which requires a relatively small diameter, in case neighbors are selected strictly according to a rank-based policy. An improved solution along the same line is to leverage the opinion of other nodes, for example, by sharing a common history. Systems based on this principle usually face two challenges. First, storage and communication is needed to store, update, and query ranking information. Implementations either use centralized servers or incur a high communication overhead by building a distributed alternative. Second, such systems are vulnerable to collusions of malicious nodes which might improve their ranking by recommending each other. Collusions can be overcome by adopting *subjective* shared histories [4, 8] where nodes favor other nodes that share a “similar opinion.” A related mechanism is the use of micro-payments; e.g. KARMA [21] uses a single currency as a way of secure trading. Problems of this approach include counterfeiting, collusions, and the inflation and deflation of the currency caused by the permanent arrival and departure of nodes.

Due to the relatively large overhead and the strict bandwidth requirements of live streaming, mechanisms based on sharing large histories with their neighbors are not viable solutions for live media dissemination. A popular and simple alternative to shared histories are *tit-for-tat* mechanisms (e.g., cf. [10]), where nodes decide *locally* whether any of its neighbors is served depending on the respective neighbor’s contribution. Clearly, one of the two par-

ties must start sending packets to prevent deadlocks and starvation and thereby risks supporting free riding nodes. Thus, the challenge of this approach is to specify when to send data blocks for free to a requesting node. Naturally, pure tit-for-tat is only feasible for protocols where packets are exchanged over bidirectional links, which immediately excludes systems based on *directed acyclic graphs*, e.g. [14], and hence also tree-based protocols. Attempts to overcome this limitation for trees usually include the splitting of streams into *stripes* [2, 20] and periodically rebuilding the trees to revert the parent-child roles. A parent can then punish a child that refused to upload in a former tree where the parent-child relation was inverse (e.g., [13]).

Because of the somewhat rigid data paths in tree-based systems, it is easier to realize incentive-compatibility in systems that do without trees, e.g., in systems such as Chainsaw [16] whose underlying topology is reminiscent of random graphs. However, it has been pointed out that using simple random topologies and a pull-based data distribution mechanism, tit-for-tat-based approaches work poorly [15]. In this paper, we show that the neighbor selection strategy is crucial, and that by carefully mixing explicit requests for missing data (*pull operations*) with *push operations* [11, 22] where data is immediately forwarded to neighboring nodes, the risk of node starvation and deadlocks can be greatly reduced. This implies that it is possible to discriminate against free riders by using simple tit-for-tat based mechanisms given a suitable network structure. The mechanism closest to ours is Swift [19], a credit-based incentive mechanism originally intended for p2p file sharing. Unfortunately, if applied directly, the Swift approach suffers from deadlocks and is not resilient to large view exploits [12, 18].

3. FAIRNESS MECHANISM

The objective of our fairness mechanism is to identify and fend off nodes in a swarm with upload rates substantially smaller than the bitrate of the data stream. We face the challenge that nodes that are eager to contribute but currently do not have to offer any data—e.g., because they have just joined the swarm—, should not be excluded.

In the following, we will give a short overview of our techniques. Our mechanism exploits some properties of live streaming which do not appear in file sharing systems. In live streaming, all nodes of a swarm are interested in the same data within the same time-frame. This mutual interest in each other’s packets facilitates pair-wise fair exchanges which are hard to achieve in file sharing systems. We employ a novel form of *time-constrained tit-for-tat exchanges* which seeks to allocate an equal share of a node’s upload bandwidth to each of its neighbors. This is in stark contrast to rank-based systems which typically strive to provide most data blocks to the best contributors among all neighboring nodes, often resulting in a skewed data distribution. Another crucial property of live streaming is that the quality degrades quickly with a larger number of buffer underflows: Nodes receiving less than roughly 80% of the packets of a live video stream are not able to watch it in reasonable quality,³ and hence free riders can be punished easily by merely reducing the stream rate to them. However, note that it is nevertheless a waste of resources to upload any data to those nodes, and it ought to be avoided.

In Section 3.1, we present the neighbor selection strategy of our mechanism. Subsequently, the incentive-compatible data distribution policy is described.

³There are video coding schemes that allow playback in reduced quality if parts of the stream are missing. However, these solutions typically employ a pyramid-like structure with multiple coding layers to gain acceptable performance, and hence our fairness mechanism can be applied on each such layer: Blocks of one layer are only useful if all blocks in lower layers are fully received.

3.1 Neighbor Selection Strategy

In order to maintain desirable swarm properties such as a small diameter, locality awareness and robustness to churn, our mechanism builds upon a hypercubic overlay network [11] as it is used in the context of *distributed hash tables* (DHT), for instance in *Passtry* [17]. Nodes maintain connections to other nodes in the overlay according to the lengths of the shared prefixes of their respective identifiers, where the node identifiers are determined, e.g., by applying a hash function to the node’s addresses in the underlying network. In particular, each node strives to maintain a connection to at least one other node whose identifier starts with the same i bits as its own identifier for each $i \geq 0$. Assuming that the hash function maps addresses to identifiers uniformly at random, the length of the longest shared prefix is bounded by $O(\log N)$ in a swarm containing N nodes, thus only $O(\log N)$ connections need to be maintained. Note that this overlay is merely used as an efficient streaming topology—no data is stored in the DHT. In our incentive mechanism, we exploit the DHT’s flexibility to choose a neighbor to replace nodes with poor upload bandwidth by better ones.

Initially, a node is assigned a random set of neighbors. Over time, a refinement process takes place as nodes learn about other nodes from their neighbors and add them to their routing table. While new nodes are selected depending on the latency measured, in order to achieve a certain *locality-awareness*, existing neighbors are continuously evaluated according to their bandwidth contributions: Each node maintains a score for each of its neighbors which represents the total number of blocks it has received from this neighbor during the last 5 seconds. All neighbors are then ranked with respect to this score. After 10 to 20 seconds, the neighbors having the worst ranks are dropped. Furthermore, each node has a small cache of nodes which have recently been replaced and whose connection requests are rejected immediately. However, these optimizations respect the condition that at least one node for each length i of the shared prefix must be retained, i.e., a node v can only be substituted if v is not the only node that shares a prefix of a specific length i . This restriction ensures that the network topology is connected and hypercubic with a bounded diameter.

Although bad nodes are dropped quickly, an unfortunate node may have many free riding neighbors in case of a large fraction of free riders. Therefore, it is vital that every node have an adequate minimum degree. In our protocol, each node has at least 25 neighbors, which works well for swarms up to 10,000 nodes.⁴

3.2 Data Distribution Policy

A key feature of our incentive mechanism is the addition of a *push-based data distribution* to the pull-based exchanges, which will be discussed afterwards. Fresh data blocks are pushed directly to a fraction of the nodes without preceding requests, in order to fuel the subsequent pull-based exchanges and to achieve low latencies in the swarm. Packets are pushed to specific neighbors depending on the prefixes of the nodes’ identifiers and the number of hops the packets have already taken. As a simple example, the source may choose up to $2^k - 1$ nodes for a small k (e.g., $k = 2$) with the property that the k -bit prefixes of the identifiers of any two nodes differ in at least one bit and are also different from the k -bit prefix of the identifier of the source. The source will then push the packet to these nodes. A node that receives such a packet proceeds accordingly, but it selects (up to $2^k - 1$) nodes whose identifiers start with the same k bits as its own identifier and whose identifiers differ in at least one bit in the next k bits of the identifier etc. It is not hard to see that no node can receive the same packet more than once.⁵

⁴Naturally, the degree will grow logarithmically as the swarm size increases.

⁵For a more detailed discussion of pushing on hypercubic overlays,

Note that while fresh data blocks are distributed on trees induced by prefixes of node identifiers, the topology is still hypercubic and every data block may be pushed along a different induced tree. What is more, only a small fraction of all data blocks are received by means of pushing operations. Therefore, our approach does not have the shortcomings of topologies consisting of one or more trees where it is inherently difficult to add a good fairness mechanism. Moreover, the hypercubic nature of the neighbor selection strategy often allows to choose among several neighbors to forward push packets, which in turn allows nodes to favor neighbors with high scores.

The majority of data blocks are received during the pulling phase by explicitly requesting them from the neighboring nodes (*pull operation*). The pull phase facilitates the adoption of a simple and practical fairness mechanism. Furthermore, explicit requests avoid redundant transmissions while being self-organizing and—at least to a certain extent—robust to dynamic changes. In our mechanism, nodes having received new data blocks notify their neighbors about the corresponding sequence numbers. A node interested in such a block then sends a request message. If multiple nodes offer the same block, the least used neighbor is selected for the download. Download slots are allocated to neighbors to avoid overloading nodes, where each neighbor receives the same amount of download slots. Both the request strategy and slot allocation are designed towards a uniform block trading with all neighbors.

The proposed algorithm combines a tit-for-tat like exchange with a certain “altruistic factor” to accommodate for newly joined nodes. Let $s_{AB}(t)$ and $r_{AB}(t)$ denote the number of packets A has sent to B and A has received from B , respectively, within the last t seconds. Let further $h_\delta(i_1, i_2, \dots, i_n)$ be a hash function that takes inputs i_1, i_2, \dots, i_n and produces a hash value in the range $\{0, \dots, \delta - 1\}$. Besides having an unused download slot available, a request from a node A to a node B for a packet with sequence number seq is granted if at least one of the following two conditions is satisfied.

Node A is allowed to request any data block from B whenever it has sent more to B than requested from B . This can be further generalized by defining a *repayment ratio* $\alpha \leq 1$ that has to be satisfied:

$$\alpha \cdot r_{AB}(t) < s_{AB}(t).$$

Naturally, either A or B has to start sending by giving the other party a certain credit $\gamma \geq 1$. This allows the other party to issue an initial request without having provided anything. In order to ensure that this cannot be exploited by selfish nodes to obtain all blocks for free by maintaining a large number of neighbors, the credit is always limited to a subset of all current blocks, determined by the address $addr_A$ of the requesting node and the sequence number seq of the requested block. For example, assuming that the hash function produces uniformly distributed hash values, the following condition limits the credit to about $\frac{1}{\delta}$ of all blocks:

$$(\alpha(r_{AB}(t) - \gamma) < s_{AB}(t)) \wedge (h_\delta(addr_A, seq) = 0).$$

It is crucial to prevent newly joined nodes willing to contribute from being excluded because they do not have anything to offer. Getting a credit and pushing new blocks to ensure that they can spread quickly is sufficient to achieve this goal as we will show in Section 4.

Choosing the *repayment ratio* α to be equal to 1 is natural as every node expects the same amount of blocks in return. The subsequent evaluation section also shows that setting the credit γ to 1 and δ to 2 results in basically no underflows for all honest nodes and that selfish nodes are never able to receive enough data blocks

the reader is referred to the pertinent literature, e.g., [11].

to view the broadcast. Of course, the quality still depends strongly on the number of disobedient nodes in the system. As we have argued before, the number of such nodes can reasonably be expected to be small, because these nodes have no incentives to stay in the swarm. Nevertheless, we will show that the fairness mechanism works reliably even if the percentage of selfish nodes is large.

Observe that while this solution successfully discriminates against selfish nodes, it is not resilient to *malicious* attacks: A malicious node can simply request the same packet again and again from different nodes. Although these repeated downloads are of no use, valuable bandwidth is wasted. Unfortunately, it seems inherently difficult to counter such an attack.

4. EVALUATION

We have performed several emulations in order to study the proposed mechanisms in different environments. First, a scenario is considered with a fraction of free riding nodes which completely refrain from uploading anything. We then study a heterogeneous swarm where some nodes have a poor Internet connection and are not able to upload a sufficient amount of blocks in order to receive the live stream. Subsequently, evidence is provided that our mechanism is robust to *large view exploits* [12, 18] and Sybil attacks [3]. Finally, our protocol is compared to other solutions. As it turns out, while our mechanism guarantees full buffers at honest nodes in the presence of free riders, other approaches inflict substantially more underflows even if there are only obedient, resource-rich nodes in the swarm, which is mainly due to the deadlocks that these approaches generate.

Unless stated otherwise, the source sends 36 data blocks per second of size 1328 bytes, resulting in a bitrate of roughly 50 KB/s. At this bitrate, codecs such as *H.264* allow the source to send high-quality video streams. Overhead induced by the various network layers, namely Ethernet, IP, UDP, and the streaming protocol itself, results in total bitrate of approximately 53 KB/s. Nodes maintain a buffer five seconds in length and the delivery is weakly synchronized with the source, that is, a block is delivered roughly five seconds after it has become available at the source. Packets are discarded after they have been delivered and are no longer available to neighbors. For most of our tests, we have used an emulated swarm of 1,000 nodes. Some tests with up to 50,000 nodes indicate that our mechanism maintains the properties observed with fewer nodes and scales well. In all our experiments, nodes do not make use of private histories, i.e., recently rejected nodes are not cached.

4.1 Free Riding Nodes

A key feature of our proposed mechanism is its ability to cope with completely free riding nodes which do not adhere to their duty of forwarding push packets or sharing packets in the pull phase. In the following, we are interested in the impact of such completely free riding nodes.

Figure 1 depicts the number of *underflows*, that is, the number of packets not delivered in time, depending on the fraction of free riding nodes. In this experiment, honest nodes have an upload bandwidth of 70 KB/s. First, the figure shows that while the free riders have a large number of underflows implying an intolerably poor playback quality, the honest nodes receive almost all packets in time. Second, it can be seen that honest nodes are hardly affected by the presence of free riders.

Figure 2 shows a snapshot of the buffers in an experiment with 20% free riding nodes. Again, we can see a clear difference between free riders and honest nodes: Most packets are available in the buffer of honest nodes, while selfish nodes suffer from many underflows.

In conclusion, while honest nodes receive almost all packets, free riders experience many gaps. For up to 15% free riders, there are

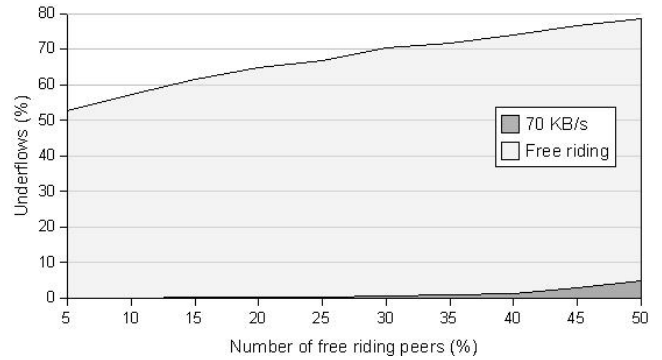


Figure 1: Emulation with different numbers of free riding nodes. Honest nodes have an upload rate of 70 KB/s, while free riding nodes do not upload anything at all.

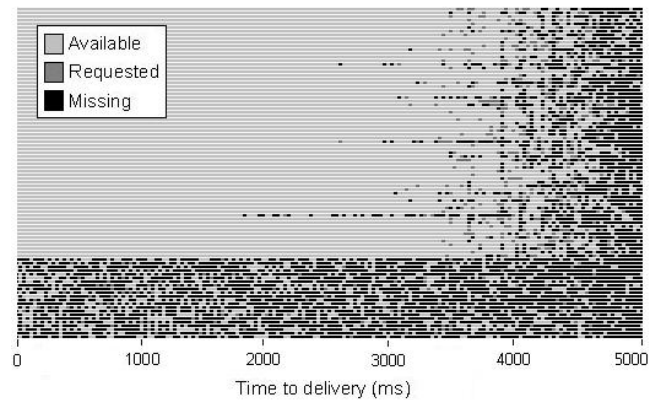


Figure 2: Snapshot of the nodes' buffers in an emulation with 20% free riding nodes (bottom). The x-axis indicates how much time is left until the data blocks have to be delivered to the application layer.

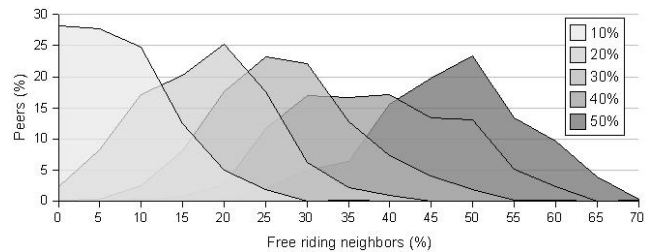


Figure 3: Percentage of free riding neighbors of honest nodes for different overall fractions of free riders.

hardly any underflows at all honest nodes, and even for up to 35% free riders underflows are limited to very few nodes. This is mainly due to the fact that free riders are not evenly distributed among the honest nodes. Of course, the higher the ratio of free riders, the more likely it is that there are nodes with many free riding neighbors (see Figure 3). In this case, a high-quality stream cannot be sustained. Fortunately, by dropping misbehaving nodes and looking for new neighbors, such nodes typically recover within seconds. Having more neighbors—which comes at a certain overhead of course—or searching more intensively for new neighbors can further alleviate the problem. With 40% and more free riders, also the bandwidth

