# LagOver: Latency Gradated Overlays

Anwitaman Datta
SCE, NTU Singapore
anwitaman@ntu.edu.sg

Ion Stoica, Mike Franklin
EECS, UC Berkeley
istoica@cs.berkeley.edu, franklin@cs.berkeley.edu

## Abstract

*We propose a new genre of overlay network for disseminating information from popular but resource constrained sources. We call this communication primitive as Latency Gradated Overlay, where information consumers self-organize themselves according to their individual resource constraints and the latency they are willing to tolerate in receiving the information from the source. Such a communication primitive finds immediate use in applications like RSS feeds aggregation. We propose heuristic algorithms to construct LagOver based on preferably some partial knowledge of the network at users (no knowledge slows the construction process) but no global coordination. The algorithms are evaluated based on simulations and show good characteristics including convergence, satisfying peers' latency and bandwidth constraints even in presence of moderately high membership dynamics. There are two points worth noting. First, optimizing jointly for latency and capacity (*i.e.*, placing nodes that have free capacity close to the source) as long as latency constraint of other nodes are not violated performs better than optimizing for latency only. The joint optimization strategy has faster convergence of the LagOver network, and can deal with adversarial workloads that optimization of only latency can not deal with. Secondly, somewhat counter-intuitively, in order to do the aforementioned joint optimization, it is sufficient to find random nodes based on only the latency constraint, since even if the capacity of individual nodes is saturated it does not matter since the LagOver network can potentially be reconfigured.*

***Keywords:*** *Overlay Networks, Peer-to-Peer, Heterogeneity, Web 2.0/RSS, Information dissemination, Algorithms*

*"If a million people subscribe to a data feed from The Boston Globe, their constant hits on the site could overwhelm our servers. What's needed is a network of feed collectors that could spread the load over a larger number of computers, the way Akamai Technology in Cambridge smooths out spikes in Internet traffic for CNN and other big firms." - Boston Globe [1]*

## 1 Introduction

Consider a large number of consumers interested in information available at a single source. Consumers typically do not have a priori knowledge about the time when new information is available at the source, and consumers have individual tolerance of the delay within which they want to receive the information.

The logical approach to adopt will be to push [6] the information from the source to the consumers whenever new information is available at the source. If the source is to deliver the new information to all the consumers directly using a source-to-each-consumer communication primitive, the system will not scale with the size of consumer population. This becomes a critical problem for popular information sources with limited resources as is increasingly the case in the age of Web 2.0, e.g., bloggers. Moreover, if users have differential time constraints, it makes the task of the source much more complicated and even unattainable particularly if the source is resource constrained. Finally, some consumers may not necessarily want to subscribe directly with the source because of privacy concerns.

A possible alternative to the client(consumer)-server(source) paradigm is to use the consumers' resources to disseminate the information in a peer-to-peer manner.

Furthermore, if different consumers have different requirements with respect to the delay to be tolerated for information delivery, then consumers can ideally be organized in the overlay such that the consumers with strictest delay constraints are placed closer to the server, and they in turn cater to other consumers with relatively relaxed delay constraints. So to say, it'd be desirable to organize the consumers in a *latency gradated overlay (LagOver)*.

In recent years RSS [13]/Atom [7] based publication and aggregation has gained immense popularity. Designed for timely delivery of new updates, RSS protocol again requires the consumers to periodically or regularly pull information from the server even if there is no new information [9] - and thus suffers from what is widely termed as the "*bandwidth overload problem*". The fundamental problem with the current realization of RSS protocol and aggregator clients is that all the clients periodically and continuously poll the source irrespective of whether there are any new updates. There are various other causes of the bandwidth overload problem, which are well summarized in [11]. Moreover the RSS protocol is increasingly being used to disseminate content, including multi-media content, which is much more resource consuming than what RSS was originally designed for. A resource constrained server will fail to cater to an increasing consumer base. RSS might very well be headed down the path of another pull based application, Pointcast, which eventually succumbed to its own popularity since frequent pulls were used to emulate push [6].

Providing a practical solution to deal with RSS's problems gives us a compelling motivation for designing a latency gradated overlay - where the consumers with strictest latency constraints are placed closest to the server and continue to pull from the server periodically, however other nodes[1] can pull from nodes upstream. Alternatively the nodes closer to the source can even push the updates downstream. Such an approach is non-intrusive to the existing infrastructure since it does not require the RSS protocol itself to be changed at all, particularly requiring no modifications at the source/servers, but needs only redesigning the clients so that the clients self-organize into a LagOver to disseminate feeds in a P2P manner.

In this paper we propose a generic communication primitive called latency gradated overlays (LagOver) which explicitly accounts for peer heterogeneity and individual preferences as inherent parts of the design. We provide heuristic algorithms to construct such latency gradated dissemination tree overlay networks in a self-organizing fashion, comprising of nodes with individual delay and bandwidth constraints. We evaluated the algorithms with extensive simulations to demonstrate the viability of LagOver even under churn (membership dynamics of the consumer population).

While we do not address the issue of incentives or altruism of individual peers and assume collaborative peers, we explicitly consider a system where individual nodes specify the maximum resource it is willing to provide, and hence the number of nodes it is willing to serve. Thus we would like to design algorithms to construct a latency gradated overlay, which not only satisfies the delay constraint of individual nodes, but also is subject to the fan-out constraint of each node. In order to facilitate the construction of such an overlay, we'd assume partial availability of global information at individual peers (Oracle based model). We shall study the influence of varying degree of global knowledge on the LagOver convergence. The assumed Oracles can be readily realized in diverse manner, and we elaborate on possible realizations when we describe the Oracle properties.

[1]We interchangeably use the terms consumers, nodes and peer. Also, we use the term "feed" loosely, and it may mean not only RSS feeds, but also any other information being disseminated in general.

## 2 Notation and system model

In Table-1 we summarize the notation used in this paper.

**Table 1. Notation**

| Notation | Meaning |
|---|---|
| $i^l_f$ | Node $i$ with maximum fanout $f$ and delay constraint $l$. |
| $f_i$ | Maximum fanout of node $i$. |
| $l_i$ | Delay constraint of node $i$. |
| Node $0$ | Feed source (server). |
| $j \leftarrow i$ | $i$ is the parent of node $j$ in the LagOver, thus $i$ pushes to $j$ or $j$ pulls from $i$. We use both $\leftarrow$ and $\rightarrow$, and only the *arrow-head* provides the context of a parent-child relation. |
| $Parent()$ | Gives a node's parent node (if any). |
| $Children()$ | Gives a set comprising a node's children. |
| $n \not\leftarrow$ | A node $n$ without parent (currently). |
| $j \not\leftarrow i$ | Node $j$ decides to leave its parent $i$. |
| $i \leftrightarrow j$ | Node $i$ interacts with node $j$. |
| $i \leftrightarrow j \leftarrow k$ | Node $i$ interacts with node $j$ s.t. $j \leftarrow k$. |
| $i \leftrightarrow j \not\leftarrow$ | Node $i$ interacts with node $j$ s.t. $j \not\leftarrow$. |
| $Root()$ | Gives the node at the root of the chain for a peer. |
| $DelayAt()$ | The actual delay perceived at a node. |

### 2.1 Assumptions and setting

#### 2.1.1 Decoupled Time

We use two notions of time - one is the interval for a node to initiate a new interaction during the bootstrapping of LagOver, other, the latency tolerated by nodes for obtaining new information. Thus LagOver will be constructed in time rounds, but this has no relation and is different from the latency constraints at each node for obtaining feed which is in effect determined by the round trip time between nodes.

#### 2.1.2 Pull at source

We consider that the information source can support only pulls from clients (as is currently for RSS). Nodes directly pull from the source at an interval of $T$ time units get information which is no staler than $T$. A server supporting push can also readily use a LagOver based P2P dissemination, but we focus here only on pull based servers.

#### 2.1.3 Local information at peers

At any given time, from the perspective of a peer $c$, lets say a chain $c \leftarrow i \leftarrow k \leftarrow ... \leftarrow r \not\leftarrow$ is formed.

We assume that the following information is available locally at all nodes, piggy-backed over the chain, which is then exploited by the nodes during their interaction with other nodes.

Any node in the chain knows the root of the chain at that instant of time, that is $Root(c) = r$. Furthermore they know their parent $Parent(c) = i$. Similarly, each node knows its direct descendants $c \in Children(i)$. If $r = 0$, that means that all the nodes in the chain are connected to the source, and hence can actually receive the feed (information).

We also assume that nodes can determine the actual maximum delay $DelayAt(c)$ in terms of overlay hops which they would observe. It is essentially determined by the length of the chain upstream plus the delay at the node closest to the source which is performing periodic pulls.

#### 2.1.4 Oracles - Partial global information

LagOver construction relies on random interaction with other nodes. While global coordination of the sequence of interactions is not necessary, partial global information in order to choose appropriate interaction partners can influence the rate and convergence of the overlay construction process. We assume that an Oracle has such information, and facilitates node interactions. The Oracle can be distinguished variously, depending on the degree of global information utilized by the Oracle while providing reference for random interactions.
***Random*:** Oracle *Random* provides to an enquiring node $i$ a random contact $j$ which is interested in the same feed. Note that in using this Oracle, no global information is being used at all, and hence determines a baseline case.
***Random-Capacity*:** Oracle *Random-Capacity* provides a random node interested in the same feed and has some free capacity (fanout not fully used) irrespective of whether the latency constraint is being satisfied.
***Random-Delay-Capacity*:** Oracle *Random-Delay-Capacity* provides a random node interested in the same feed and can satisfy latency constraint of the querying peer (that is, its actual observed delay is less than the querying peer's latency constraint) and also has free capacity.
***Random-Delay*:** Oracle *Random-Delay* provides any random node interested in the same feed whose actual latency is less than the enquiring peer's latency constraint, irrespective of whether it has any free capacity (unused fanout).

The above mentioned Oracles may be realized in various manner. For instance, if nodes participate in an unstructured network, random walkers can be used to implement Oracle *Random*. The other Oracles *Random-Delay-and/or-Latency* require a directory service, which may be provided by a centralized authority like Syndic8 [12], but can also be realized if the nodes organize as a distributed hash table. More realistically, we can assume a separate open service like (and even using) OpenDHT [10] to provide the service of the Oracle - since it is run in a single trust domain using a relatively stable and dedicated infrastructure like PlanetLab and has proven to support various other services.

Realizing the Oracle using more stable and smaller population of peers (logically different from the consumers) is also desirable, since the consumer population itself will be highly volatile and can be potentially large, such that additionally maintaining a DHT for all these peers put together may not be feasible.

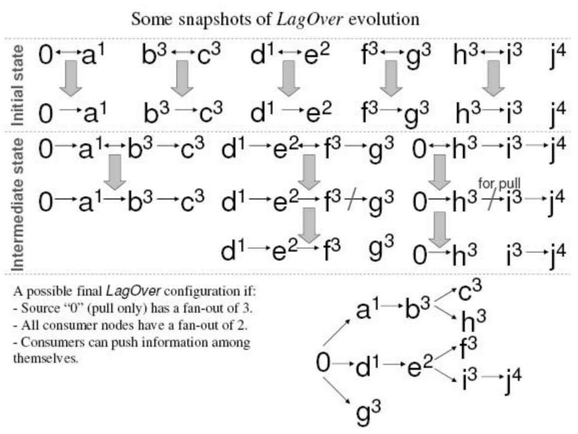
## 3 Bootsrapping a LagOver

The basic idea behind the latency gradated overlay construction involves random interaction among nodes in order to locally decide which of these nodes should be closer to the source in comparison to others. Thus, during the construction process, several disjoint groups will be formed. Eventually, all these groups should coalesce into a dissemination tree with the source (node 0) as its root, where ideally the distance of each node from the source is such that its latency constraint is satisfied, while respecting all the nodes' fanout constraints.

Depending upon the bilateral action of nodes during the random interactions, and the specific properties (determined by the Oracle) of the random peers with which these interactions are performed, the interesting aspects of the LagOver construction algorithm include the convergence of the isolated nodes to form a single-source dissemination tree and resilience of the overlay construction and maintenance process against churn.

### 3.1 Greedy algorithm

A natural approach to satisfy the delay constraints of all nodes in the system would be to place the nodes in the tree according to their delay constraints, *i.e.*, place the nodes with tighter delay constraints closer to the source. The idea is intuitive and the algorithmic details (omitted here because of space constraints) can be found in an extended version [5]. The more robust hybrid construction algorithm is later explained in full detail. To summarize, the principal ideas of the greedy algorithm include:
- Peer interactions are facilitated by the Oracle, but also by the peers themselves. If peer $i$ interacts with peer $j$ which already has a parent node, then if $l_j < l_i$, $i$ tries to become a child node of $j$, possibly by becoming parent of one of $j$'s current children $m$ provided $m$'s latency constraint is not violated by the reconfiguration. Unless node $i$ finds a suitable parent, it is referred to $k$ - parent of node $j$, which is further upstream and more likely to fulfill $i$'s latency constraint.
- Opportunistic cluster formation of peers based on their relative delay constraints, with the peers with strictest delay constraints directly pulling information from the source.
- Reconfiguration of the clusters (maintenance operations) upon encountering peers with stricter delay constraints.



A possible final *LagOver* configuration if:
- Source "0" (pull only) has a fan-out of 3.
- All consumer nodes have a fan-out of 2.
- Consumers can push information among themselves.

**Figure 1. Evolution of a LagOver**

Reconfiguration is necessary because of two reasons: (i) The nodes are not synchronized, and a node with laxer latency constraint may still contact the server earlier than a node with stricter latency constraint. (ii) Churn in the system means that a node with stricter latency constraint may (re-)join the system after some node with laxer constraint has already started receiving information directly from the source, and needs to be replaced.

The greedy algorithm ensures that if $i \leftarrow j$ for any $i$ and $j$, then $l_j \leq l_i$.

Next we illustrate with some intermediate snapshots of a toy example system how the greedy LagOver construction works, particularly to show that the greedy construction algorithm needs to be complemented with a maintenance strategy in order to successfully construct a LagOver.

## 3.2 Maintenance operations

Consider a system comprising of the source $0_3$ and consumers $a_2^1, b_2^3, c_2^3, d_2^1, e_2^2, f_2^3, g_2^3, h_2^3, i_2^3, j_2^4$. The time evolution of a possible $LagOver$ based on random interaction of nodes is partially shown in Figure 1.

At any intermediate stage, there are separate groups of nodes forming parts of the dissemination tree optimistically. Ideally, they all should eventually coalesce into a single connected group and act as a dissemination tree. Since these groups are formed opportunistically, it may so happen that at a later point of time some of the nodes discover that their latency constraint can not be met if they continue to stay connected in the same manner. This necessitates that these nodes discard their current parents and try to find an alternate node which can meet the nodes' latency constraint. In the example, this is the case for nodes $g$ and $i$ after the second step. Once the disconnection actions $g \not\leftarrow f$ and $i \not\leftarrow h$ are undertaken they reach the state $g \not\leftarrow$ and $i \not\leftarrow$, and thus locally restarts the LagOver construction algorithm, that is to say, tries to locate and interact with other suitable nodes. Note that though $c \leftarrow b$ seems to be a similar case, but since $b \leftarrow a$ and $a$ pulls from source such that it has the latest updates within a delay of 1 time unit, $b$ gets the updates pushed by $a$ within a delay of 2 time units and similarly $c$ gets it within 3 time units. So to say, $c \leftarrow b \leftarrow a$ is a configuration that meets the latency constraint of all the concerned nodes and needs no maintenance operations.

While it is necessary to locally detect that a node's latency constraint can't be met, if all such nodes immediately discard their parent nodes, it will not only waste a lot of the past interactions and the structure built therefrom, but also such a knee-jerk reactive mechanism will cause a larger than necessary dynamicity of the evolving LagOver and require even more future interactions. For instance, in this example the configuration $j \leftarrow i$ can still be reused once $i$ discovers a suitable parent node. Thus a pragmatic decision needs to be taken by nodes to discard their parent nodes based on local information.

For the Greedy LagOver construction a node $i$ can determine if $DelayAt(i) = l_i + 1$ and $Root(i) = 0$ to discard its parent node. We summarized the maintenance operation in Algorithm 1.

***Lemma:*** *The maintenance strategy (of Algorithm 1) is sufficient for the greedy construction algorithm.*

***Proof:*** Only the node in a chain which is the first (upstream) one with its latency constraint violated need to leave the chain. Nodes downstream do not need to do any thing. Ofcourse, at a later time, after other reconfigurations, they too may have to actuate the maintenance action. We only need to prove that, for the first node in a chain with latency constraint violation, $DelayAt(i) = l_i + 1$ and $Root(i) = 0$.

**Algorithm 1** Maintenance operations at node $i$

```
1: if $i \leftarrow j$ s.t. ($DelayAt(i) = l_i + 1$ and $Root(i) = 0$) then
2:     $i \not\leftarrow j$ {The condition is sufficient for greedily constructed overlay,
       but a more aggressive condition (timeout based) is required for the
       hybrid construction (Algorithm 2) described later.}
3: end if
```

Consider $i \leftarrow j$. Recall that the greedy algorithm ensures that $l_j \leq l_i$. Thus, two cases arise. If $l_j < l_i$ and $j$'s latency constraint is met, then $i$'s latency constraint will also be met. Hence the violation may occur only if $l_j = l_i$. In that case, if $i$ is the first node in the chain with its latency constraint violated $DelayAt(i) \geq l_i$ then necessarily $DelayAt(j) \leq l_j$. If $DelayAt(j) < l_j$ then $i$'s latency is not violated either, hence $DelayAt(j) = l_j$. Since $l_j = l_i$, thus it follows that $DelayAt(i) = l_i + 1$. — *q.e.d.*

## 3.3 A sufficient condition for existence of LagOver

Lets say $N_l$ represent the set of nodes with latency constraint of $l$, where $l \geq 1$.

***Observation:*** If $\sum_{p \in N_l} f_p \geq |N_{l+1}|$, and if the latency constraint of all $p \in N_l$ is satisfied, then the latency constraint of all $p \in N_{l+1}$ can also be satisfied.

***Lemma:*** *If latency constraint for all nodes with latency constraints $< l$ can be met, then it can also be met for all nodes with constraint $l$ if* $|N_l| \leq \sum_{p \in N_{l-1}} f_p + \sum_{l' < l-1} (\sum_{l'} f_p - |N_{l'+1}|)$

This yields a lower-bound for the maximum number of nodes with a specific latency constraint $l$ that can be accommodated in a LagOver, thus a sufficient condition for existence of a LagOver for a given population of nodes.

### 3.3.1 Sufficient but not necessary: A counter-example

Note that the sufficiency condition is not a necessary condition for existence of a LagOver, however the greedy algorithm is not well disposed to explore alternative possibilities when the sufficiency condition is not met.

We use another toy example to show that there are settings where it is possible to meet the latency and fanout constraints of all nodes, but the greedy algorithm simply can not achieve the desirable configuration of nodes to do so.

Consider the set of nodes: $\{0_1, 1_1^1, 2_1^2, 3_2^4, 4_1^3, 5_0^3\}$, where $0_1$ means that the source will directly support only 1 consumer. For these nodes, one possible configuration of the LagOver is: $5_0^3 \leftarrow 3_2^4, 4_1^3 \leftarrow 3_2^4, 3_2^4 \leftarrow 2_1^2, 2_1^2 \leftarrow 1_1^1, 1_1^1 \leftarrow 0_1$. The greedy algorithm however will not establish the configurations $5_0^3 \leftarrow 3_2^4, 4_1^3 \leftarrow 3_2^4$, and instead will try to put nodes 4 or 5 closer to the source than node 3, in which case it will not be possible to meet all the nodes' latency and fanout constraints.

## 3.4 Hybrid algorithm

Though such adversarial situations may not be commonplace, it nonetheless is interesting to investigate if it is at all possible to devise an algorithm which tries to explore other configurations, by not always greedily pushing nodes with stricter latency constraints closer to the source, but doing so only if otherwise the node's latency constraint is violated, but in general giving preference to nodes with higher fanout, so that more nodes can be accommodated downstream. Note that a greedy preference of only fanout would have worked best in keeping the dissemination tree depth least and minimizing the achieved average latency if there were no individual and diverse latency constraints. However, in order to accommodate both the individual latency constraints as well as to serve larger number of nodes downstream, we need a hybrid strategy.

Algorithm 2 is such a hybrid LagOver construction mechanism. While this algorithm is expected to have more flexibility since it can explore a greater combination of node configurations even in an adversarial situation, if the sufficiency condition is not met the peers may still not converge to a possible LagOver, even if such a configuration exists.

**Algorithm 2** Hybrid LagOver construction algorithm as executed at node $i$

```
while $i \not\leftarrow$ do
  if Timeout = True then
    if Free capacity at 0 (source) then
      $i \leftarrow 0$
    else
      if $\exists\, c \leftarrow 0$ s.t. $l_c > l_i$ then
        $c \leftarrow i \leftarrow 0$ {Needed primarily because of churn}
      else
        Reset counter for Timeout
      end if
    end if
  else
    $i \leftrightarrow j$ {interact with some j referred at last interaction or by Oracle. There is an exception: It may happen that the Oracle finds no suitable j, and the peer needs to wait and try again. }
    if j=0 then
      Same as when contacting server because of timeout.
    else
      if $j \not\leftarrow$ then
        if $i$ or $j$ has unused fanout then
          try $i \leftarrow j$ or $j \leftarrow i$
          {Give preference to the node with larger fanout to be parent if fanout is available at both nodes.
          If $f_i = f_j$, give preference to the node with stricter latency constraint to be the parent node.}
        end if
      else
        if $j \leftarrow 0$ then
          if Pull only server then
            if $l_i < l_j$ then
              try $j \leftarrow i \leftarrow 0$
            else
              try $i \leftarrow j$ or else try $l \leftarrow i \leftarrow j$
              {Refer $i$ to 0 otherwise. - *Need to check what's being done for $l_i < l_j$.*}
            end if
          else
            if $f_i > f_j$ then
              try $j \leftarrow i \leftarrow 0$ {$i$ may need to discard one child node. }
            else
              try $i \leftarrow j$ or else try $m \leftarrow i \leftarrow j$
            end if
          end if
        else
          if $f_i \geq f_j$ then
            try $j \leftarrow i \leftarrow k$ {$i$ may discard one of its current children}
          else
            try if possible $i \leftarrow j$ or else try $m \leftarrow i \leftarrow j$ where $m \in Children(j)$ s.t. the reconfiguration does not violate $m$'s latency constraint.
          end if
          If neither of the above configurations are possible because $DelayAt(j) \geq l_i$, $i$ will use $k$ as its next reference, trying to move closer to the server. Otherwise $i$ will refer to the Oracle.
        end if
      end if
    end if
  end if
end while
```

The interaction of a node at the server is the same as in the case of the greedy algorithm. Thus nodes with strictest latency constraint communicate directly with the source. This is necessary particularly for the case where the source supports only content pulls.

For a $i \leftrightarrow j \not\leftarrow$ interaction initiated by $i$, unlike the greedy case where preference is given to latency constraint, the hybrid mechanism prefers nodes with higher fanout to be the parent node (*steps 9-11*). This however may sometimes lead to a situation where the node at the tip of such a group eventually contacts the server, and the nodes downstream have their latency constraint violated, whence some of these downstream nodes leave this node, and continues with the construction process, potentially leading to later reconfiguration like $j \leftarrow i \leftarrow 0$ (*step 17*). However note that in a system of large peer population, only a small fraction of nodes have to go through such a reconfiguration, and in the absence of global knowledge, particularly about which nodes really have strictest latency constraints, it is necessary to use the opportunistic configurations of *steps 9-11*.

Upon $i$ interacting with $j \leftarrow 0$, if the source is pushing information then any node can potentially be placed communicating directly with the server, and thus the node with larger fanout is typically given preference (*steps 21-23*). Latency thus plays a deciding role only if latency constraint is violated in one of the two possible parent-child configurations or else when fanout of both interacting nodes is the same (*steps 24-25*). For a source supporting only pulls, latency based reconfiguration (*step 17*) may be necessary. The case $i \leftrightarrow j \leftarrow k$ is dealt with in *steps 28-34*, where reconfigurations are done on the basis of nodes' fanouts.

Since the hybrid algorithm does not order the LagOver strictly based on individual nodes' latency constraints, but opts for arbitrary configurations as long as the constraints are met, it has a greater flexibility. However, unlike the greedy algorithm, $i \leftarrow j$ for any $i$ and $j$ provides no information about the relative ordering of $l_i$ and $l_j$. This has its implications for the maintenance algorithm (Lemma of Section 3.2). Hence, apart from using $DelayAt(i) = l_i + 1$ and $Root(i) = 0$ at any time, a more aggressive manner of discarding parent node is necessary. We still need to dampen the $i \not\leftarrow j$ events for the same reasons as elaborated for the greedy algorithm. So if $DelayAt(i) > l_i$ and $Root(i) = 0$, to reduce knee-jerk reactions node $i$ waits for a (maintenance) timeout before it leaves its parent node.

## 4 Simulation settings

Both the proposed LagOver construction algorithms were evaluated using a discrete time simulator. Simulations were run for various workloads, primarily characterized by the nodes' latency constraints, fanouts and the churn in the system. Unless otherwise mentioned, we implicitly assume that the nodes originally meet the sufficiency condition (Section 3.3) of existence of a LagOver. However, under churn, the set of online peers at any given time may violate this condition.

### 4.1 Topological constraints

For the latency and fanout constraints, we use several combinations of inputs. Since the constraints restrict the potential topologies of the resulting LagOver, we call them the topological constraints.

***Use full available capacity** ($Tf1$)**:** A simple case is where all nodes have same fanout, and the latency constraints are chosen such that nodes utilize the whole capacity of the system. Consider a scenario where the source supports 3 consumers, and each consumer in turn supports 3 more consumers and there are 3, 9, 27, 81 (and so on) consumers with latency constraints 1, 2, 3, 4 units (and so on) respectively. Then the complete capacity available at nodes upstream is required to satisfy the nodes downstream.

***Random delay and latency constraints** ($Rand$)**:** In this scenario nodes have random delay and capacity constraints, and the delays and capacities are not correlated.

***Bimodal correlated constraints** ($BiCorr$)**:** A worst case scenario is where peers with strict latency constraint also have low downstream bandwidth capacity. We furthermore assume that peers have either low (modem users) or rather abundant (broadband users) bandwidth capacity. In typical simulation runs peers had latency constraints such that it could be anywhere between 1 to 10 time units, while a fanout of either 1 or 2 (low) or 7 or 8 (high) subject to the condition that peers with latency constraint less than 3 time units also had low fanout.

***Bimodal uncorrelated constraints** ($BiUnCorr$)**:** A contrasting scenario is where peers have bimodal bandwidth constraints but the latency constraint is uncorrelated with bandwidth constraint. Here, there is no systematic conflict of interest in putting these peers close to the server.

## 5 Evaluation

The algorithms are evaluated via simulations and show good characteristics, including convergence fulfilling peers' latency and bandwidth constraints even in presence of moderately high membership dynamics. The construction algorithms are evaluated based on *construction latency*, *i.e.*, the time needed to complete the LagOver construction process.

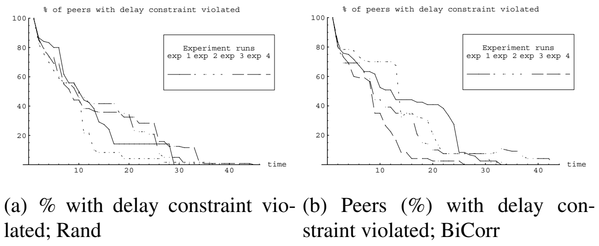
There are two points worth noting. First, optimizing jointly for latency and capacity (*i.e.*, placing nodes with free capacity close to the source) performs better than optimizing for latency only (greedy algorithm), as long as latency constraint of other nodes are not violated, as done in the hybrid algorithm . The joint optimization strategy has faster convergence of the LagOver network than optimizing with respect to latency alone, and can deal with adversarial workloads that optimization of only latency can not deal with.

Secondly, somewhat counter-intuitively, in order to do the aforementioned joint optimization, it is sufficient to find random nodes based on only the latency constraint (Oracle *Random-Delay*). Indeed, even the capacity saturation of individual nodes does not matter, since the LagOver network can potentially be reconfigured. Using this partial information significantly improves the performance with respect to the scenario where no information is used (Oracle *Random*).

Using more information, based on both delay and availability of unused fanout (Oracle *Random-Delay-Capacity*) may at a first glance seem to be a smarter strategy, but this disallows interactions where reconfigurations are possible, seriously impeding the convergence. Depending on the workload, the performance can be even worse than the case where no information is used.

## 5.1 Variation in the rate of convergence

For the same workload (topological constraint, peer population and choice of oracle), each variant of the LagOver construction algorithm has a high variation in the time required to converge. This is shown in Figure 2 for the execution of the Greedy algorithm using Oracle *Random-Delay* for various workloads. Similar behavior was observed for other Oracles for both the algorithms.



(a) % with delay constraint violated; Rand (b) Peers (%) with delay constraint violated; BiCorr

**Figure 2. Variation in convergence of greedy algorithm (using Oracle *Random-Delay*) without churn**

Thus, for the rest of the paper, typically experiments were repeated 5 times and the median performance was chosen as the representative for the specific experiment setting.

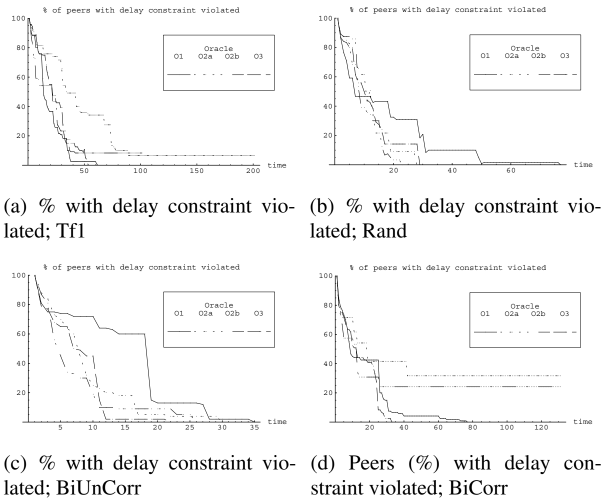
## 5.2 Impact of Oracles on the convergence

We conduct simulations with 120 peers in each case for $Tf1$, $Rand$, $BiUnCorr$ and $BiUnCorr$ topological constraints under no churn, in order to determine the impact of Oracles on the (greedy) algorithm's performance. We observe in Figure 3 that Oracle *Random-Delay* has the best performance in many of the settings, and has good performance overall. Similar behavior of better performance using Oracle *Random-Delay* was observed for experiments conducted with the Hybrid LagOver construction algorithm. The behavior of the different Oracles, and the practical implications may be explained as follows.

Oracle *Random* provides any random peer, which may however not satisfy the latency constraint, nor have free capacity. Note that this is the case where no global information is being used. For diverse workloads, the LagOver converges but incurs longer latencies.

Oracle *Random-Capacity* would, at a first glance seem to provide a suitable match since it provides a peer with free capacity. *Random-Delay-Capacity* provides even more precise matches, with peers also satisfying the latency constraint. While this may seem to help - in fact it can be detrimental, because it may so happen that at a time instant there is no more peer left which has free capacity and would satisfy time constraint, but only reconfigurations can lead to addition of new peers in the LagOver. Which is why using *Random-Capacity* and *Random-Delay-Capacity* often not only take long time, but sometimes simply does not converge, since they fail to provide any interaction partner at all. This shows that misusing global information may in fact even be counter productive.

In contrast, Oracle *Random-Delay* does not care about whether fanout of a peer is fully used or not, and provides any peer which will satisfy the latency constraint. Thus, during the interactions, reconfigurations may be possible. As shown by our simulation results, Oracle *Random-Delay* not only achieves better performance, but also converges in the absence of churn and as long as the sufficiency criterion (Section 3.3) is met. For the rest of the experiments we use Oracle *Random-Delay*.



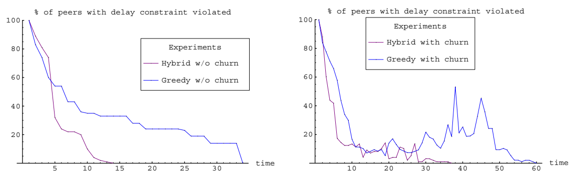
(a) % with delay constraint violated; Tf1 (b) % with delay constraint violated; Rand

(c) % with delay constraint violated; BiUnCorr (d) Peers (%) with delay constraint violated; BiCorr

**Figure 3. Greedy algorithm performance for various Oracles and topological constraints without churn. Oracles:** *Random (O1)*, *Random-Capacity (O2a)*, *Random-Delay-Capacity (O2b)*, *Random-Delay (O3)*.

## 5.3 Construction under churn

In P2P networks some peers leave while some (re-)join. This is called churn. We study the dynamic resilience of the system in the presence of churn, comparing the construction algorithms. For the experiments, it is assumed that initially all peers are online. In each time step, online peers leave the network with a probability 0.01, while offline peers re-join with a probability 0.2. Figure 4 shows the performance of the algorithms without and with churn for peers with correlated bimodal latency and fanout constraints.



(a) Without churn (b) With churn

**Figure 4. Greedy vs. Hybrid algorithm: Bimodal correlated latency and fanout**

We again observe that both without and under churn, for various workloads, the Hybrid algorithm outperforms the Greedy algorithm. These results suggest that the Hybrid algorithm is effective even in the absence of prior knowledge of the workload and environment.

In real life, synchronization of peer interactions is unrealistic. We conducted further experiments where peers interacted asynchronously, *i.e.* different peers need different amount of time to complete the interactions. Asynchrony slowed down the overlay construction, but interestingly did not affect the eventual convergence to a LagOver [5].

## 6 Related work

There is a huge body of work focusing on various aspects of content distribution and QoS multicast routing [3]. But most of these do not address the specific problems arising in RSS style systems - including source supporting pull based content access, irregular and small content updates, and diverse user constraints or preferences of tolerating delay in being notified the updates. dissemination of small updates occurring at possibly unpredictable times is also different from typical distribution problem addressed by BitTorrent [4] and other networks.

The FeedTree [11] approach proposes feed dissemination using P2P communication among feed consumers. It is built on top of the Scribe [2] system. It assumes consumers of all different feeds participate in a single distributed hash table (DHT) based overlay network. This underlying DHT is then exploited to discover other peers interested in the same feed. Our use of Oracle, possibly using a DHT service is on similar lines. In FeedTree, an overlay multicast tree per feed is established on top of the DHT overlay. It thus needs to deal with churn of both the underlying DHT as well as the multicast trees for feeds, apart involving peers uninterested in a feed in multicasting the same.

## 7 Conclusion

We introduced the concept of a latency gradated overlay (LagOver) network - a novel genre of dissemination tree which meets individual user's differential latency and fanout constraints. We also proposed heuristic algorithms to construct such an overlay without coordination, using easily obtainable partial global information. We evaluate the algorithms based on simulations. Though the design of LagOver was motivated by RSS content dissemination, because of the generic treatment we hope that there will be other elastic real-time applications which will find the properties of LagOver useful, and use it as a substrate. One promising application is that of peer-to-peer video delivery based on multipath routing [8], where each peer participates in multiple LagOvers with different time constraints - one LagOver for each of the multiple paths. In the presented work one LagOver is established to disseminate content from one source. Reusing part of the LagOver for multiple sources by exploiting intersecting consumers, as well as building the LagOver based on locality contexts, like clients within same domain, ISP or timezone forming the overlay may substantially improve the global performance and resource usage, and are interesting potential future work.

## References

[1] H. Bray. Time for the next (really) big Internet idea. `http://www.boston.com/business/personaltech/articles/2005/07/04/time_for_the_next_really_big_internet_idea`.

[2] M. Castro, P. Druschel, A-M. Kermarrec, and A. Rowstron. Scribe: A large-scale and decentralised application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications (JSAC)*, 2002.

[3] S. Chen and K. Nahrstedt. An overview of quality-of-service routing for the next generation high-speed networks: Problems and solutions. *IEEE Network Magazine, Special Issue on Transmission and Distribution of Digital Video, 1998.*, 1998.

[4] B. Cohen. Incentives build robustness in bittorrent. In *Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems*, 2003.

[5] A. Datta, I. Stoica, and M. Franklin. LagOver: Latency Gradated Overlays (Extended version of ICDCS'07). In *http://www.ntu.edu.sg/home/anwitaman/ResearchLagOver.html*.

[6] M. Franklin and S. Zdonik. "Data in your face": Push technology in perspective. In *SIGMOD*, 1998.

[7] IETF. Atom Publishing Format and Protocol. `http://www.ietf.org/html.charters/atompub-charter.html`.

[8] C. Jiancong, S.-H.G. Chan, and V.O.K. Li. Multipath routing for video delivery over bandwidth-limited networks. *IEEE Journal on Selected Areas in Communications (JSAC)*, Dec., 2004.

[9] H. Liu, V. Ramasubramanian, and E. G. Sirer. Client Behavior and Feed Characteristics of RSS, A Publish-Subscribe System for Web Micronews. In *Proceedings of Internet Measurement Conference (IMC)*, 2005.

[10] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. OpenDHT: A Public DHT Service and Its Uses. In *SIGCOMM*, 2005.

[11] D. Sandler, A. Mislove, A. Post, and P. Druschel. FeedTree: Sharing Web micronews with peer-to-peer event notification. In *IPTPS*, 2005.

[12] Syndic8. A directory of nearly all available XML and RSS driven weblogs and syndicated newsfeeds. `http://www.syndic8.com/`.

[13] D. Winer. Really Simple Syndication. `http://blogs.law.harvard.edu/tech/rss`.