Dynamic Random Broadcast Trees for Improved Load-balancing

Sérgio Duarte, José Legatheaux, Margarida Mamede and Nuno Preguiça

CITI / Departamento de Informática - Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa,
Quinta da Torre, 2829 -516 Caparica, Portugal
Email:{smd, jose.legatheaux, mm, nmp}@di.fct.unl.pt

Abstract. Load-balancing is an important issue in P2P dissemination trees. This paper addresses this problem with an algorithm for creating random broadcast trees, on-the-fly, in a decentralized manner. Moreover, by allowing each node to choose its fanout degree dynamically, the algorithm is capable of biasing the load of each node taking into account multiple sources of traffic. The experimental results provided, in particular those regarding its use as an integral part of an one-hop DHT, show that it offers clear load-balancing improvements compared to regular trees.

1 Introduction

The Internet is today an established medium for mass distribution of information and content. However, despite its ubiquity and reach, dissemination in the Internet is essentially achieved by sending the same information many times over and separately to each user. This poses a problem of efficiency, which stems from the lack in the current Internet of native multipoint communication services at the network-level. There are several reasons for this, as discussed in [7], but despite the significant number of proposals made over the years [11], the fact remains that IP Multicast has yet to leave its niche status.

Overlay networks overcome some of the limitations of the Internet protocol stack by implementing routing at the application level. The complete freedom afforded by this approach has led to the appearance of an extensive body of work regarding application-level multipoint communication [10,13,**?**,1,3,4,8]. Excluding epidemic diffusion [4,**?**], which is a form of mitigated flooding; a common characteristic these approaches share is the use of dissemination trees to deliver information to the intended recipient nodes.

One important issue that arises regarding the use of dissemination trees is that of load-balancing [1]. For any given tree, a large fraction of the nodes is composed of terminal nodes that play little or no part in the dissemination process. One way to counter this unfairness is to split the content over several well-chosen dissemination trees, while ensuring that each node is interior node on one of them (and a terminal one on the rest). If the right proportion is maintained, the net incoming and outgoing traffic can be balanced [1,14,**?**].

In this paper, we address the problem of load-balancing broadcast trees, more specifically in the context of Distributed Hash Tables (DHTs) that keep significant routing tables at each node, such as [5,6]. This sort of DHT foregoes table compactness for the ability to perform lookups and routing in just one or two hops, as opposed to mainstream DHTs [**?**,12,**?**,**?**]. To keep these large routing tables fresh, each node arrival has to be broadcasted to the whole system or to a sizable partition. This can be costly for high churn rates but for some applications, such as cloud computing **??**, routing in as few as possible hops can be important because it can significantly improve system response.

The broadcasting technique described in this paper relies on random trees built on-the-fly for each individual broadcast message. The algorithm used to build such broadcast trees ensures that the probability of a node being chosen for interior node is such that, over time, it will

cancel the times it is picked for terminal node (when it does nothing). Furthermore, the tree building algorithm lets nodes vary independently their respective workload by tweaking the tree fanout degree from message to message. This provides for a measure of fine control that allows other sources of traffic to be taken into consideration when measuring fairness and setting the load-balancing goals. This algorithm can be exploited or adapted to drive the broadcasting component of DHTs that optimize routing for constant hops. The main contributions of this paper are:

- to present random dissemination trees as the basis for load-balancing;
- an algorithm to build random dissemination trees on-the-fly;
- an evaluation of the proposed algorithm in static and dynamic settings.

The rest of the paper is structured as follows. Section 2 describes the solution proposed. Section 3 provides an analysis of the proposed algorithm, followed by its experimental evaluation in Section 4. The paper concludes a review of related work in Section 5 and some final remarks and future work in Section 6.

2 DyRT - Dynamic Random Trees

2.1 Goals and Assumptions

The algorithm, named DyRT, assumes a P2P network environment with full connectivity among all pairs of nodes. Its purpose is to support message broadcasting among all participant peers, allowing for multiple concurrent sources and variable payload sizes. The goal is to achieve that in manner that is *fair*, a condition that for each node is met when it leaves the system and the net difference between the total upload and download traffic processed is negligible. DyRT is not meant to be fault-tolerant per se, in the sense that its intended service is best-effort; nor is it expected to produce broadcast trees that minimize latency by exploiting proximity metrics, although it can be modified to handle the latter.

The algorithm does not assume any prior network organization, either structured or unstructured, in the form of pre-established links. Logically, connections are established as needed and destroyed afterwards. In practice, some can be cached and re-used with positive effects, although this is out of the scope of this paper.

The key assumption is that DyRT assumes that every node has access to a database containing all its intended recipients. How this database is obtained is a problem that can be partially solved by DyRT itself. Using DyRT to broadcast new node arrivals (optionally node departures) into the system, allows these databases to be populated, in a manner that is akin to the workings of one-hop DHTs, such as [5]. However, any inconsistencies among the node's databases will tend to undermine performance and lead to missed deliveries. In an one-hop DHT usage scenario, these will degrade lookup performance, adding hops for the individual

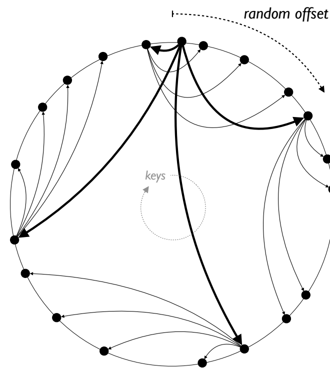


Fig. 1. Example of a DyRT broadcast tree for a fanout degree of 4

Listing 1 The DyRT algorithm in pseudo code

```
    -doBroadcast( Payload<...>)
        r ← random()
(1)     Range^0 ← [r ··· 2^128[ + [0 ··· r[
        send Broadcast<self.key, Range^0, Payload<...>) to self

    ↳onReceive( Broadcast<root.key, Range^i, Payload<...>> )
        send Payload<...> to self
        if self.key ≠ root.key
(a)         Range^{i+1} ← Range^i[self.key + 1 ···]
        else
            Range^{i+1} ← Range^i
        fanout ← getFanout()
        remaining ← DB.nodes( Range^i )
(2)     if remaining.size ⩽ fanout
            foreach node in remaining
                if node.key ≠ root.key
                    send Payload<...> to node
                        ↳ onFailure
                            remove node from DB.nodes
        else
(3)         subranges ← {Range_1^{i+1}, Range_2^{i+1}, ..., Range_fanout^{i+1}}
            foreach Range_j^{i+1} in subranges
                candidates ← DB.nodes(Range_j^{i+1})
(4)             foreach node in candidates
                    if node.key ≠ root.key
                        send Broadcast<root.key, Range_j^{i+1}, Payload<...>> to node
                            ↳ onSuccess
                                break
                            ↳ onFailure
                                remove node from DB.nodes

    ↳onReceive( Payload<...>)
        ...
```

keys affected. Therefore, some additional means to ensure that node databases converge over time may be required depending on the actual application and how damaging missed deliveries are. In this regard, we have explored a gossip-based solution and found that it can fulfill that role; its discussion is out of the scope of this paper, but it has no impact on the load-balancing performance of the proposed solution.

2.2 Broadcasting Algorithm

Each DyRT node requires an unique key, which can be computed by secure hashing its endpoint data (IP + port) and preserving enough bits to ensure uniqueness with high probability, for instance 128 bits. The key space is treated as directed, ordered and circular, using modular arithmetic. The algorithm works by recursively subdividing the key space, on-the-fly, for each message individually. Thus, at any given point of the broadcast tree, the nodes that still remain to be contacted, are represented by a pair of keys, i.e., a *range*. At the root, the initial range covers the entire key space (1), and gets progressively subdivided, according to the desired fanout, as the message is passed from a node to its children down the tree. Besides de payload, the broadcast message carries the key of the root node and the key range that remains to be processed; the heather overhead is thus small. Sending the root key along prevents the root node from participating twice in the broadcast.

Listing 2 Dynamic fanout computation and update based on current traffic ratio

```
-getFanout()
    adjustment ← 1−(upload_total - download_total)/(1 + download_total)
    targetFanout ← min(MAX_BROADCAST_FANOUT, max(1.5, targetFanout × adjustment ))
    if random() > targetFanout - floor(targetFanout)
        return floor(targetFanout)
    else
        return ceil(targetFanout)
-init()
    targetFanout ← MAX_BROADCAST_FANOUT
```

Upon reception of a broadcast message, the recipient node must determine if the range of keys that remains can be subdivided further, which happens when the actual number of nodes found in the database is smaller than the fanout degree (2). If that is the case, its children will be terminal nodes and only need to receive the payload. Otherwise, the node has to continue the recursion and divide the range into disjoint subranges, according to the desired fanout degree (3). This can be accomplished in key space[1]. However, for better load-balancing, it is best to subdivide a range according to the actual nodes that remain. But, rather, than generating equal-sized partitions in terms of the number of nodes[2], better results are achieved if the boundaries are chosen so that each resulting sub-range denotes a complete subtree, except may be one. The node with the "smallest" key in the range is selected to continue the recursion. If the selected node fails to respond then the next one in "increasing" key order is tried, until one replies or the range is exhausted[3]. As a result, subranges shrink without fragmentation and can be represented in a compact way. [4] Finally, a different tree is produced for each broadcast message by rotating the initial key range by a random offset, so that range subdivision is not always aligned on the same boundaries (1), as also illustrated in Fig. 1.

This algorithm allows each node considerable freedom regarding the choice tree fanout degree and how subranges are produces. It only requires that the resulting subranges get narrower and are disjoint. This enables us to use a dynamic fanout at each node to bias the its load according to some metric. As an example, Listing 2 shows how the total node traffic can be used to increase or decrease the target fanout degree dynamically. The most interesting aspect of this is that other sources of traffic (e.g., unicast or other trees) can be also taken into account, provided as a whole everything balances out.

3 Analysis

Our analysis will consider a simplified static setting, i.e., without *churn*, composed of a constant number of N nodes. The goal is to obtain an estimate of the communication costs, in terms of *upload* and *download* capacities that each participating node has to invest to support a continuous sequence of broadcasts. To that end, we analize the cost of broadcasting a message of size m_t bytes at a rate of r messages per second. Moreover, we assume that N is such that the dissemination tree is complete for a given fanout f. So, if h is the height of the dissemination tree, N satisfies the following expression:

$$N = \sum_{i=0}^{h} f^i = \frac{f^{h+1} - 1}{f - 1} \tag{1}$$

[1] Only acceptable for the top levels of the tree as an computational optimization. As ranges get narrower, any non-uniformity in the node distribution will get more prevalent and impact negatively on load-balancing.

[2] Produces broadcasting trees with ragged bottom levels

[3] Note that "smallest key" and "increasing key order" needs to be interpreted taking into account the directed and circular nature of the key space.

[4] However, any nodes unknown yet to the parent whose keys are smaller than the selected children will be missed (a). It is possible to mitigate this, by having the child node sweep the part of the subrange prior to its key.

Since messages are broadcasted using a random tree for each message, the overall cost of dissemination for any given node will depend of its position in each tree. When chance makes it a leaf node, it only has to download m_t bytes. When it is chosen for interior node, the node also has to relay to its children $f.m_t$ bytes of upload capacity.

As the number of disseminated messages accumulate, the average communication rates will depend on the probability of a node being a leaf or interior node. Given that there will be f^h leaf nodes in the broadcast tree, using (1) we derive these expressions:

$$\begin{aligned} P(leaf) &= \frac{f^h}{N} = \frac{f^h(f-1)}{f^{h+1}-1} = \frac{f^{h+1}-f^h}{f^{h+1}-1} \approx \frac{f^{h+1}-f^h}{f^{h+1}} = 1 - \frac{1}{f} \\ P(interior) &= \frac{1}{f} \end{aligned} \tag{2}$$

Thus, the average *upstream* (b_u) and *downstream* (b_d) capacities required at each node are:

$$\begin{aligned} b_u &= P(interior) \times f.m_t \times r = \frac{1}{f} \times f.m_t \times r = m_t.r \\ b_d &= m_t.r \end{aligned} \tag{3}$$

Meaning that under true random conditions, treating all nodes fairly, we can expect the average upstream and downstream traffic in each node to converge to the same average value. A value that only depends on the dissemination rate and the size of the messages. In particular, this individual cost does not depend on the number of nodes that compose the dissemination tree.

To assess and compare load-balancing performance among solutions and parametrizations, we can use the standard deviation of the average *upstream* network usage, (b_u), as the metric indicator. The higher the standard deviation, the less balanced is the work distribution among the broadcast tree nodes. Using (1), (2) and (3) and from the definition of standard deviation, the following relations can be derived, again, for a complete broadcast tree of N nodes and an uniform fanout of f.

$$\begin{aligned} stdev(b_u) &= \sqrt{\frac{1}{N} \times \left[\sum_{interior\ nodes} (b_u - f \times m_t.r)^2 + \sum_{terminal\ nodes} (b_u - 0 \times m_t.r)^2 \right]} \\ &\approx \sqrt{\frac{1}{N} \times \left[N\frac{1}{f}(b_u - f \times b_u)^2 + N(1-\frac{1}{f})(b_u - 0 \times b_u)^2 \right]} \\ &= \sqrt{\frac{1}{f}{b_u}^2(1-f)^2 + (1-\frac{1}{f}){b_u}^2} = b_u\sqrt{\frac{1}{f}(1-f)^2 + (1-\frac{1}{f})} = b_u\sqrt{f-1} \\ \frac{stdev(b_u)}{b_u} &= \sqrt{f-1} \end{aligned} \tag{4}$$

4 Evaluation

We evaluated DyRT under two main scenarios. The first set of experiments consisted in implementing DyRT in a custom discreve event simulator, and running it as a standalone algorithm to test it under different configurations. These experiments involved static network conditions and intended to provide a direct comparison with the analytical results shown in the previous section.

A second set of more elaborate experiments concerned a fully dynamic network. These are meant to assess the expected performance of the algorithm under real usage settings. To that end, we implemented an one-hop DHT, denoted Catadupa from now on, that uses DyRT to broadcast node arrivals and node departures to all participant nodes. Since DyRT is not itself fault tolerant, Catadupa also includes complementary mechanisms to ensure eventual consistency of the databases of the participating nodes. The three parametrizations of DyRT evaluated are discussed in the following paragraphs.

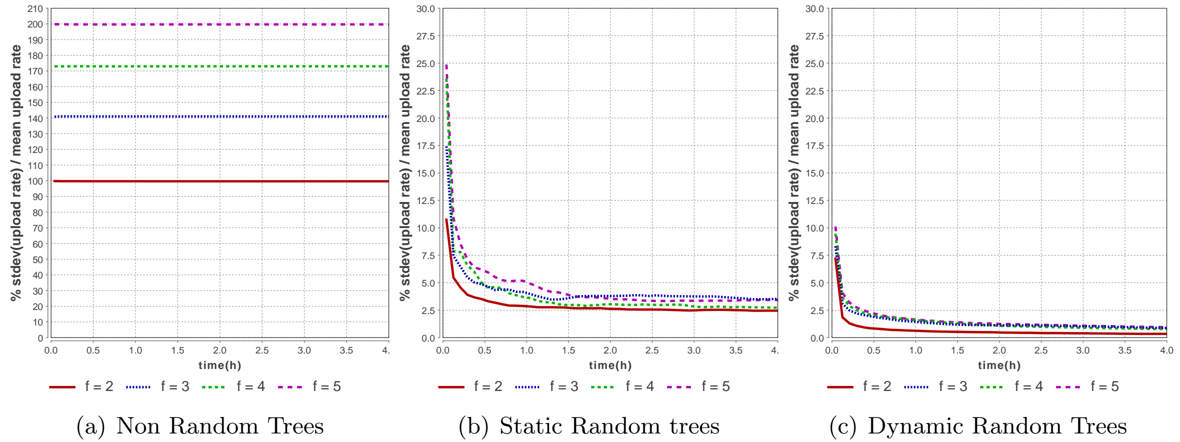
(A) Non Random Trees To produce non random trees, DyRT was parametrized so that the initial range of keys of the broadcast message always starts at key 0, instead of using a random offset. In a static network environment this produces always the same tree, for a given root node. For a randomly selected root node, the trees are not exactly the same, but share large subtrees, corresponding to the first level partitions extraneous to the root node.

(B) Static Random Trees This is the basic configuration of DyRT. Random Trees are produced by using a different random offset for each message for the initial key range. The recursive range subdivision uses the same fanout degree in every node.

(C) Dynamic Random Trees In this configuration DyRT allows each node to adapt its fanout degree according a given traffic ratio, increasing or decreasing it fractionally to even out the upload and download rates. In this particular case, traffic used to compute the ratio refers to the broadcast process alone (, since there are no other sources of communication to balance).

4.1 Static Broadcast Scenario

In this set of experiments, DyRT was used to broadcast messages, at a rate of 1 per second, in a network composed of 10,000 nodes. Regarding the choice for the root of the tree, two modalities were tried: (i) picking a random node for the root of the tree for each message; (ii) using the same node for the root of the tree for every message. The two produced the same results when rounded to units of percentage, therefore, we only present the results obtained for the former (i). Changing the size of the messages, or using messages of random size, did not show significant differences, either, so the results presented are for messages of 1000 bytes. The statistics collected correspond to a broadcast session lasting 4 hours (14,400 broadcasts). Fanouts degrees ranging from 2 to 5 were evaluated.



(a) Non Random Trees (b) Static Random trees (c) Dynamic Random Trees

Fig. 2. Summary of results for the Static Broadcast Scenario, showing standard deviation of the upload rate as a percentage of the average, for a static network of 10,000 nodes

Discussion The results obtained are summarized in the charts of Fig. 4.1, where the standard deviation of the upload rate is shown as a percentage of the average. *Non Random Trees*, c.f. Fig. 2(a), produce results matching very closely the analysis made in section 3, regarding the expression 4 that relates the standard deviation with the square root of the fanout minus one. When DyRT is configured to produce a random tree for each individual message, scenario, c.f. Fig. 2(b), there is a sharp decrease in the standard deviation of the average upload rate of the nodes to less than 5% of the average in all the fanout configurations tested. Furthermore, if in addition to the use of random trees, DyRT is allowed to vary the fanout degree independently for each node, c.f. Fig. 2(c), then the standard deviation drops quickly to less than 2.5% of the average in all configurations. However, this improvement comes at the expense of an increase

in broadcast latency, both in terms of average and standard deviation (jitter). This is to be expected, since when a node decreases its fanout degree, it causes its broadcast subtree to increase in height and, consequently, in latency.

These results clearly confirm that the use of random trees can improve load-balancing quite significantly. Allowing each node to dynamically determine its fanout degree, provides further improvement but the degradation it causes on message latency needs to be factored in. However, the purpose of using dynamic fanout degrees is to allow other sources of traffic to be taken into account and achieve load-balancing for the whole traffic exchanged by a participant node. This is discussed in more detail next, in the context of a dynamic network scenario.

4.2 Dynamic Broadcast Scenarios

This part of the evaluation pertains to the performance of DyRT algorithm in the presence of network dynamism, close to its usage scenario. For this, we implemented an one-hop DHT, denoted Catadupa from now on, that uses DyRT to broadcast node arrivals and node departures to all participant nodes. Since DyRT is not itself fault tolerant, Catadupa also includes a complementary repair mechanism to ensure eventual consistency of the node databases of the participating nodes. Furthermore, as an additional source of traffic, Catadupa nodes also need to perform sporadic point-to-point upload sessions to other nodes as they join the Catadupa network[5]. Due to the size of these uploads, they impact negatively on load-balancing; the use of dynamic fanouts in DyRT is meant to address and mitigate the problem.

All communication is supported over (simulated) connection-oriented TCP streams. And, traffic accounting includes the size of the TCP/IP headers for connection establishment and teardown. Moreover,the upload capacity used in Catadupa is the same for all nodes, with 25 KB/s and a peak rate of 50 KB/s allocated for DyRT[6]. As a result, messages experience delays due both to network latency and the capped upload links of the nodes. Download capacity was not limited.

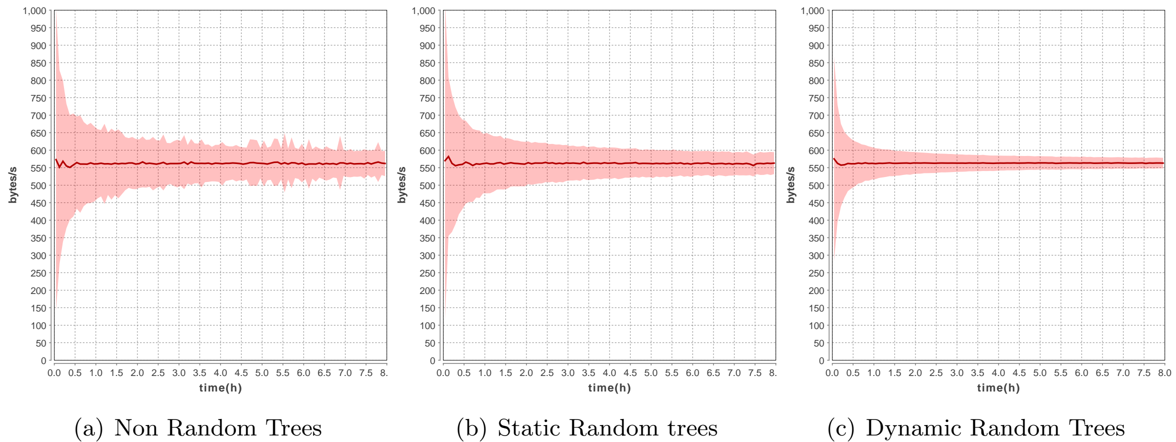
To model system dynamics, we adopted a churn model that consisted of an exponential distribution to generate inter-node arrival times, while node session times are obtained from a Weibull distribution with shape parameter of 1.8 and mean duration of 4 hours, truncated at a maximum of 8 hours. End of session events are treated as fail-stop node failures.

The first set of results compares the load-balancing performance of DyRT while fully embedded in Catadupa, and in a setting akin to a large data center (cluster). For these experiments, the node average arrival rate used was 2 events per second, which, upon reaching equilibrium, produces a system that hoovers at around the 26,000 nodes mark. Moreover, node arrivals and departures are broadcasted at a rate of 3 concurrent broadcasts every 30 seconds, with variable payload sizes and using trees with a maximum fanout degree of 4. Traffic statistics were recorded upon node failure (due to end of session events) and were binned according to session duration in sets of 300 seconds.

Discussion Figure 4.2 summarizes the results obtained for the three types of broadcasting trees discussed previously, labeled earlier as (A), (B) and (C). The charts show average rates as thick lines and the shaded areas correspond to one standard deviation above and below of said average. It is clear that *Dynamic Random Trees*, c.f. Fig. 3(c), produces the smallest variation of the upload rate, denoting the best overall load-balancing performance. *Random Trees with Static Fanout*, see Fig. 3(b), is not as good, showing a wider variation of the upload rate among the nodes. As for *Non Random Trees*, Fig. 3(a), it is prone to spikes and is still the worst option, despite showing improved results compared to the static scenario. This improvement is explained

[5] Essentially, each fresh node has to acquire a copy of the current membership state of the Catadupa network. It does so by downloading pieces of it from several nodes already in the system.

[6] The actual average traffic rates used are a small fraction of this figure.



(a) Non Random Trees (b) Static Random trees (c) Dynamic Random Trees

Fig. 3. Average upload rates obtained using DyRT as the broadcasting algorithm of Catadupa, after 22 hours of simulated time and a network of around 26,000 nodes. The shading denotes $\pm$ 1 standard deviation from the average

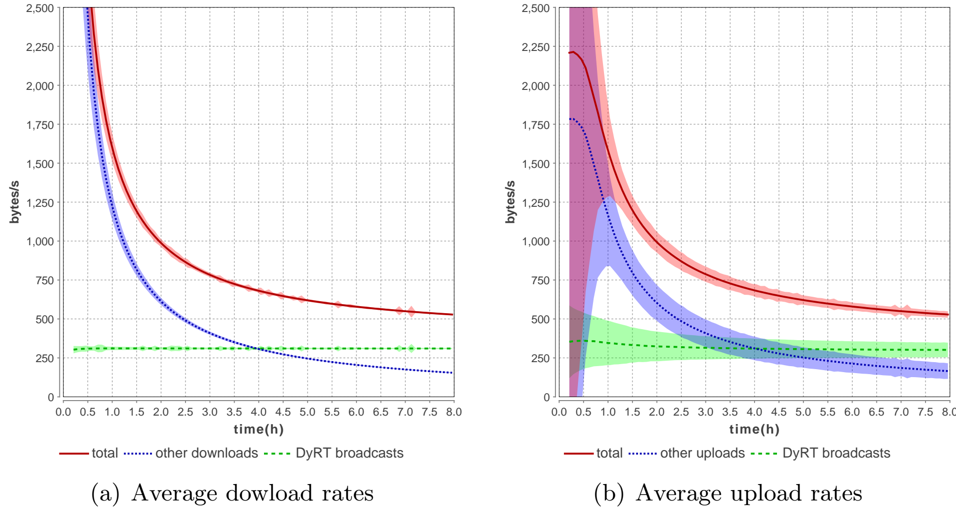
by the effect of network churn in the tree topology; the random events that continously add and remove of nodes lead to trees that slowly change over time. The effect, however, is not able to prevent the spikes in the statistics, which are caused by nodes whose keys are the smallest in each of the key ranges/partitions that denote the top levels of the broadcast tree. Without the random offset, such nodes are always picked for interior nodes, unless they fail or new nodes appear with even smaller keys, which gets harder as the keys get closer to lower limit of the partitions in question. The three types of tree show improved load-balancing as node sessions get longer, since these nodes accumulate more time for randomness to take effect.

We performed one final experiment, summarized by Fig. 4.2, which comprises another Catadupa network, with an average node arrival rate of 8 events per second, which stabilizes at around 105,000 nodes, pushing the setting towards a cloud computing environment. DyRT was configured to use dynamic fanouts ranging between 2 to 4. The statistics account 55 hours of simulated time, i.e., several generations of nodes, also collected in 300 seconds bins. Besides the higher churn rate, this Catadupa version also requires its nodes to download particularly large databases from other participant nodes when they arrive and, consequently, during their stay, they have to support comparable uploads for the benefit of other nodes. Two main observations can be made from the two charts. (1) The average total upload rate matches very closely the average total download rate, especially for sessions longer than 1.5 hours. Thus, the longer nodes stay in the system the more fairly they are treated. (2) The standard deviation of the average total is smaller than the standard deviation of each of its compomentes, meaning that the components are correlated and cancel each other to some extent. The source of correlation is the use of dynamic fanouts degrees in DyRT based on the total traffic each node has processed.

5 Related Work

In [5], the authors proposed the so-called one-hop routing DHT and were among the first to observe that the cost of fully replicating the membership of a large system is not proportional to the system size, but to the membership churn rate. For membership replication, they assemble broadcast trees anchored on specific nodes, leading to as much as 10x of upload rate imbalance among the participating nodes. In a similar setting, DyRT is able to achieve even loads for all nodes that stay in the system for a reasonable amount of time.

Pastel [2] is an extension to Pastry DHT[12] to allow routing in a constant number of hops. A broadcast primitive is used to populate full membership state at each node and also to provide an application-level broadcast service. It exploits the structured nature inherited from Pastry and assembles the broadcast tree based on node id prefixes. The resulting tree is not balanced, with nodes having as many as log(N) children. If the same application broadcast root node is



(a) Average dowload rates

(b) Average upload rates

Fig. 4. Summary of results for using DyRT as the broadcasting algorithm of Catadupa, for 55 hours of simulated time and a network of around 105,000 nodes. The shading represents ± 3 standard deviations from the average

repeatedly used, the tree does not change much, exacerbating the problem. In DyRT, on the other hand, the fanout degree is bounded, which is important when participating nodes have limited upload links.

Application-level broadcasting is a problem also inherent to P2P live streaming or video on demand applications **??**. The popularity of structured overlays [12,**?**,**?**] and their promises of scalability made them primary candidates for implementing those services. Many of the mainstream DHTs can be adapted to support multicast or broadcast primitives [13,3], but the resulting trees suffer from poor load balancing properties [1]. as the resulting fat trees can have unbounded out degrees and because a large fraction of the peers do no useful work. The randomization technique used by DyRT does not map well to small state DHTs and cannot be used to solve the load-balancing problem in those systems. Splitstream [1] proposes splitting content over multiple disjoint trees, built on top of Pastry, so that any node will act as a data relay in at least one of them. DyRT uses a simple randomization technique to the same end, except that it does so for every message on-the-fly. DyRT can handle payloads of variable sizes, whereas a static multi-tree approach suffers in such cases, unless large payloads can be split evenly among all the trees. Unstructured overlays have also been explored to support P2P multicast services, ChunkySpread [14] is an example that has been designed with load-balancing in mind. It, too, breaks content into chucks that are sent along multiple trees. Tree formation involves searching for suitable partner nodes in a random graph. Unlike DyRT, its load-balancing requires probing other nodes, which encourages the use of long-lived trees.

There are several gossip-based forms of application-level broadcasting or multicast [4,8] do without the use trees, arguing that they are fragile and costly to keep.

[9]

6 Final remarks and Future Work

In this paper we have presented DyRT, a broadcasting algorithm based on random trees with dynamic out degree, which can be used to improve load-balancing in one-hop DHTs, such as [5]. The experimental results provided, in particular those concerning the use of the algorithm to broadcast node arrivals and departures in an one-dht, do support that claim. It is also been shown that building random trees on-the-fly with dynamic fanouts, has clear advantages to using trees with a fixed fanout or non-random trees for the same purpose. Furthermore, it

allows load-balancing to take into account other sources of traffic extraneous to the broadcast stream.

As for future work, we are pursuing two directions. Firstly, we want to adapt DyRT to reuse TCP connections for multiple broadcasts, without compromising its load-balancing properties. This will address the overhead incurred with connection setup and teardown and is easily achieved by not changing the initial random offset on a per broadcast basis. However, how long each tree can be maintained without ill effects is still an open issue. Secondly, we want to modify DyRT to drive other forms of Catadupa that only keep track of a fraction of the entire system membership. The main issue we foresee is that incomplete information available at each node may lead to the formation cycles, at the top levels of the tree. To counter that, we are evaluating bloomfilters as a compact way to store the path to the root of the tree in each message and thus detect and avoid possible routing cycles.

References

1. M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth multicast in cooperative environments. In *The 19th ACM Symposium on Operating System Principles (SOSP 2003)*, October 2003.
2. N. Cruces, R. Rodrigues, and P. Ferreira. Pastel: Bridging the gap between structured and large-state overlays. In *CCGRID '08: Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid*, pages 49–57, Washington, DC, USA, 2008. IEEE Computer Society.
3. S. El-Ansary, L. O. Alima, P. Brand, and S. Haridi. Efficient broadcast in structured p2p networks,. In *2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, February 2003.
4. P. Eugster, S. Handurukande, R. Guerraoui, A. Kermarrec, and P. Kouznetsov. Lightweight probabilistic broadcast. In *The International Conference on Dependable Systems and Networks (DNS 2001)*, 2001.
5. A. Gupta, B. Liskov, and R. Rodrigues. One hop lookups for peer-to-peer overlays. In *Ninth Workshop on Hot Topics in Operating Systems (HotOS-IX)*, pages 7–12, Lihue, Hawaii, May 2003.
6. A. Gupta, B. Liskov, and R. Rodrigues. Efficient routing for peer-to-peer overlays. In *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, pages 9–9, Berkeley, CA, USA, 2004. USENIX Association.
7. H. Holbrook and D. Cheriton. IP multicast channels: EXPRESS support for large-scale single-source applications. *ACM SIGCOMM Computer Communication Review*, 29(4):65–78, 1999.
8. A.-M. Kermarrec, L. Massoulié, and A. J. Ganesh. Probabilistic reliable dissemination in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(3):248–258, 2003.
9. B. Leong and J. Li. Achieving One-Hop DHT Lookup and Strong Stabilization by Passing Tokens. In H. K. Pung, Francis, C. K. Tham, and S. Kuttan, editors, *Proceedings of the 12th International Conference on Networks 2004 (ICON 2004)*, volume 1, pages 344–350, Piscataway, NJ, USA, 2004. IEEE Press.
10. D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: An application level multicast infrastructure. In *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems (USITS '01)*, pages 49–60, San Francisco, CA, USA, Mar. 2001.
11. M. Ramalho. Intra- and inter-domain multicast routing protocols: A survey and taxonomy. *IEEE Communications Surveys & Tutorials*, 3(1):2–25, March 2000.
12. A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350, London, UK, November 2001. Springer-Verlag.
13. A. I. T. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. SCRIBE: The design of a large-scale event notification infrastructure. In *Networked Group Communication*, pages 30–43, 2001.
14. V. Venkataraman, K. Yoshida, and P. Francis. Chunkyspread: Heterogeneous unstructured tree-based peer-to-peer multicast. In *ICNP '06: Proceedings of the Proceedings of the 2006 IEEE International Conference on Network Protocols*, pages 2–11, Washington, DC, USA, 2006. IEEE Computer Society.