Centaur: Dynamic Message Dissemination over Online Social Networks

Shen Li *, Lu Su†, Yerzhan Suleimenov*, Hengchang Liu‡, Tarek Abdelzaher*, Guihai Chen§
* University of Illinois at Urbana-Champaign, USA 61801 † State University of New York at Buffalo, USA 14260
‡ University of Science and Technology of China, China 215123 § Shanghai Jiaotong University, China 200240
{shenli3, suleime1, zaher}@illinois.edu, lusu@baffulo.edu, hcliu@ustc.edu.cn, gchen@cs.sjtu.edu.cn

***Abstract*—We present the design, implementation, and evaluation of Centaur, an application-level user-assisted message dissemination solution for Online Social Networks (OSN). Characteristics of OSNs make their message dissemination distinct from scenarios like multicast streaming and P2P file sharing. First, updates issued by each user are sporadic and the "online" follower set is highly dynamic. Hence, it is unnecessarily expensive to maintain always-alive multicast topologies. Second, the key advantage of OSNs over traditional media is realtime update, which would be greatly shadowed if it takes long to construct well-shaped dissemination structures. Therefore, in contrast to the multitude of prior multicast solutions, Centaur constructs location-aware dissemination trees locally for each incoming message. We implement a prototype with Cirrus and evaluate it with Twitter data. Experiment results show that Centaur achieves 98% delivery ratio and few seconds of delay with only around one tenth server traffic compared to centralized solutions used in many current OSNs.**

***Index Terms*—online social networks; message dissemination; approximate algorithms; data centers; fault tolerance.**

I. Introduction

Online Social Networks (OSNs) connect people by delivering one user's updates to all of her followers and friends. With this basic service model, different application scenarios have been proposed, ranging from capturing the trends of the world to detecting earthquakes [1, 2]. With the growth of OSN population, it becomes increasingly difficult to maintain the message delivering service with low latency and high reliability [3]. For example, Lady Gaga, a popular and active user on Twitter, has more than 41 million followers on Twitter. Even if only 1% of them are online, each of her new tweets would still need to be delivered to hundreds of thousands of online users in realtime [4]. Lady Gaga is only one user. According to the data crawled from Twitter, aggregately, servers need to send out more than 700 thousand messages every second on average, consuming huge amount of network bandwidth. To mitigate this problem, we propose to amortize the communication overhead to a large number of online users using server-constructed *per-message* dissemination trees.

Three OSN characteristics distinguish its message dissemination from earlier systems, such as multicast video streaming and P2P file sharing. First, most users spend only a very short period of time on OSNs each day. For example, statistics show that, on average, each user spends about 405 minutes on Facebook every month [4]. This behavior results in a highly dynamic online user set, where one user is considered "online" if she is using the OSN software, or browsing the OSN web page. Therefore, user "sessions" are more fleeting than typical media multicast. Second, updates issued by any individual user are very sporadic. For instance, at the time of writing, Lady Gaga has posted around 2700 tweets since she registered in March 2008. On average, she tweets less than twice per day. Each tweet is a much shorter update than a typical file or video download. Thus, it would be unnecessarily expensive to dynamically maintain an always-alive dissemination structure over the online receiver set. Third, the key advantage of OSNs over traditional media is realtime update, which would be undermined if it takes a long time to converge on a well-shaped dissemination structure. (More comparisons are discussed in Section II.) Accordingly, the contribution of this paper lies in efficient *server-side* construction of *per-message* dissemination trees.

Server-side construction of per-message tree makes sense for several reasons. Given the dynamic and sporadic features of OSN users, a user is not likely to receive multiple updates from the same followee within one online session. Therefore, different messages are expected to be delivered to very different online receiver sets. In this case, every update calls for a new dissemination tree. As OSN servers already have global knowledge of every online user, and are always the source of message dissemination, they are naturally the right entities to carry out the tree construction algorithm. In seeking practicality, the dissemination tree construction algorithm needs to address the following four challenges:

- Minimize the message delivery latency: Realtime updating, as a key advantage of OSNs over traditional media, has to be considered and maintained when constructing the dissemination tree.
- Bound the overhead on user side: The number of children of a node in the dissemination tree indicates the number of message copies the node needs to send out. This number should be small enough, so that the cost of assisting message dissemination is negligible for users.
- Achieve high message delivery rate: User nodes are unreliable. They may fail (*e.g.*, crash or leave) at any point in time, resulting in message loss to all descendants in the dissemination tree. Thus, the solution needs to deal with failures properly and achieve high delivery ratio.
- Reduce the overhead on the server side: Given that popular OSNs own a huge user base [4] and receive a large number of updates every second [5], the algorithm has to be computationally efficient to allow the server to compute one new tree for each new update.

Centaur addresses all four challenges by actively pushing updates to users and carefully designing a server-side per-message dissemination tree construction algorithm, which we call Randomized **L**ocation **A**ware **M**essage **D**issemination algorithm (LAMDR). By considering the location information, LAMDR is able to prioritize geographically short connections over long connections. Low user overhead and high message delivery rate are thus achieved by setting an upper bound on node degree and tree height and taking advantage of redundancy. The tree construction algorithm has a guaranteed approximation ratio, and runs in $\mathcal{O}(n \log n)$ time, where n is the number of online followers of the message publisher.

978-1-4799-3572-7/14/$31.00 ©2014 IEEE

Technically, users may choose to opt in or out by configuring their profiles. For example, a mobile phone user with limited 3G data plan is likely to decline. Centaur focus on reducing server-side overhead for OSN message delivery, leaving privacy and security for future work.

Centaur consists of two parts: (1) Centaur servers that keep track of the online user set, and run the tree construction algorithm, and (2) Centaur web pages that allow users to receive messages from Centaur servers or other users, and disseminate these received messages according to the given tree structure. Please note, Centaur handles only message disseminations, and does not modify the persistent data storage systems. Offline users still get historical data from OSN servers upon their next logins. Our experimental results show that, with Centaur, the server is able to save more than one order of magnitude on communication, and each user spends only a few KB bandwidth during peak time. At the same time, it takes only a few seconds to disseminate any message, with a delivery rate over 98%.

The remainder of this paper is organized as follows. We survey the related work in Section II. Section III describes the system model and problem formulation. Then we show system design details in Section IV, and Section V. Implementation and evaluation are presented in Section VI and Section VII, respectively. Finally, Section VIII concludes the paper.

II. Related Work

In this section we describe the differences between OSN message dissemination and three earlier scenarios, namely, multicast streaming, publish/subscribe systems, and other P2P systems [6–14]. In each comparison, we also explain the reason why existing solutions are not applicable to OSNs.

Overlay network multicast has been discussed for years in applications such as media video streaming, teleconferences, file sharing, etc. Brosh *et al.* [7] design a multicast tree construction algorithm which directly maps load to delay penalty. Recently, literatures [8, 15] discuss and analyze hybrid cloud-P2P content distribution systems for applications like YouTube. As the streaming video or teleconference usually lasts relatively long, maintaining an always-alive structure is reasonable and useful. However, in the OSN scenario, every user is a publisher (channel), owns a unique receiver set, and sporadically issues new updates. Hence, the cost of maintaining an always-alive multicast topology far exceeds the gain of disseminating a few sporadic updates.

Another related topic is publish/subscribe (Pub/Sub) systems. Much of the effort has been spent on elaborately organizing brokers or event servers. Brokers are usually third-party servers that connect publishers and subscribers. Corona [9] employs a set of intermediate brokers to keep track of users' interest and aggregates polls towards target web pages. SIENA [10] makes use of a set of dedicated servers to form an event notification network, which is transparent to end users. In their design, end users only publish or receive data, but do not act as data relays. Publiy+ [11] is a hybrid solution that combines (Pub/Sub) brokers with user formed P2P networks. Their design offloads the overhead of disseminating huge volumes of data from brokers to subscriber themselves. Nevertheless, in OSNs, dedicated broker servers do not exist. According to our evaluation results, OSNs do not need dedicated brokers either, since performance can be greatly improved by intelligently making use of user side resources.

Location-aware solutions have been discussed for many years in both unstructured and structured P2P systems. Liu *et al.* [12] try to improve the performance of unstructured P2P systems with location-aware topology matching. Peers cut off low productive connections and prefer physically closer neighbors. Qiu *et al.* [13] propose a Peer-exchange Routing Optimization Protocol (PROP). In this protocol, two peers exchange all or part of their neighbors to switch from geographically longer links to shorter ones. In Literature [14], the authors also utilize biased neighbor selection, such that, each peer chooses the majority, but not all, of their neighbors within the same ISP. However, without a centralized server that has a global view, it takes time for each peer to discover enough information, which impairs the realtime properties of OSNs. Besides, as we argued before, it does not make sense to maintain an always-alive topology due to highly sporadic user updates. The cost of maintaining the topology will be much larger than the gain of disseminating the messages. Instead, we explore the viability of constructing a new topology for every new message.

By taking sporadic, dynamic, and realtime properties into consideration, we design Centaur, a server-side per-message tree construction and dissemination solution for OSNs, which preserves the aforementioned three important properties, and at the same time significantly reduces the server side communication overhead.

III. System Model and Problem Formulation

In the system model section, we describe notations and assumptions. Objectives and problem abstraction are discussed in the problem formulation section.

A. System Model

For user i, let $\mathcal{F}_i$ denote his or her follower set. Assume $\mathcal{U}^t$ is the online user set at time t, the server only needs to notify online followers when receiving updates from the followee. For offline followers, they will be notified next time when they login to the OSN application. Set $\mathcal{F}_i$ changes relatively slowly and mostly incrementally, while set $\mathcal{U}^t$ is highly dynamic, as each user might stay online for only a few minutes to check updates [4]. Let $\mathcal{M}_i^j$ denote the j^{th} message published by user i at time t. Given the sporadic nature of user updates, user i publishes at most one message in each time slot. Let $\mathcal{R}_i^j$ denote the receiver set of $\mathcal{M}_i^j$, which is the intersection of $\mathcal{F}_i$ and $\mathcal{U}^t$ (*i.e*, $\mathcal{R}_i^j = \mathcal{F}_i \cap \mathcal{U}^t$). Given that $\mathcal{U}^t$ is highly dynamic, $\mathcal{R}_i^{j+1}$ and $\mathcal{R}_i^j$ may be completely different sets. Hence, a different dissemination tree is required for each new message.

In most of current implementations, OSN servers are responsible for notifying every online follower. However, handling all the workload on server side leads to considerable service rejection rate and response latency [3]. We propose to utilize user side resources to assist message dissemination and improve the service performance. In order to take latency into

consideration, we need an estimation of message forwarding delay between users. Given the large number of users, it is too costly to keep track of the RTT between every two users in realtime. Instead, we use geographic distance between two users to approximate the Round Trip Time (RTT) [16, 17]. This geographic information is available as the server knows each user's IP address, which can in turn be translated into location information [18].

B. Problem Formulation

Let $V = \{v_0, v_1, ..., v_n\}$ denote the node (receivers) set, and r denote the root node. Let $v_i.x$ and $v_i.y$ denote the x and y coordinates of vertex v_i, which are derived by applying Mercator projection [19] to users' GPS locations. All nodes form a complete graph, where the length of edge (v_i, v_j) is the Euclidean distance between the two nodes. In the dissemination tree, let d_i denote the number of children of v_i. We further define two types of tree height for the dissemination tree: hop height and distance height. For v_i, hop height (h_i) is the number of edges along the path from r to v_i, while distance height (H_i) is the length of the path.

When developing message dissemination algorithms for OSNs, we are concerned with following requirements:

- The message delivery delay should be minimized, which can be achieved by minimizing $\max_i\{H_i\}$.
- The overhead added to each participating user should be negligibly low. In order to meet this objective, we can set an upper bound on d_i when constructing the dissemination tree.
- The message delivery rate should be high enough. One user node is able to receive a message if all of its ascendants on the dissemination tree behave properly. Assume users are equally probable to fail, then, nodes with smaller hop heights are more likely to receive the message as they have fewer ascendants.

Taking above concerns into account, we formally formulate the problem as follows: given the set V of n nodes, the root node r, the upper bound on the number of children d, and hop height upper bound h, the algorithm needs to construct a dissemination tree to cover all vertices in V exactly once, such that the maximum H_i is minimized. We call the induced graph a LAMD tree, which is short for **L**ocation-**A**ware **M**essage **D**issemination Tree. Constructing a LAMD tree is obviously an NP-hard problem, as the $(d = 1, h = n)$ solution is a Hamilton path. Therefore, we look for good approximations.

As OSN servers see large number of updates every second [5], the computational complexity of the tree construction algorithm has to be low enough, so that the server can afford the cost to construct a new tree for each incoming message. When designing algorithms, we consider only the single-server case, which is the basic building block for more complicated cases. In scenarios where multiple geographically distributed servers are in effect, the service provider may first map the user's IP address to one or more servers, and each server then performs our algorithm separately.

IV. APPROXIMATE LAMD TREE

For each message, all followers form a complete graph on the overlay network. Computing the edge set costs $\mathcal{O}(n^2)$ computational steps, which is already too expensive as popular users may own millions of online followers. To avoid the huge computational overhead, we present Approximate LAMD Tree Algorithm (LAMDA) that partitions the node set into smaller groups, in each of which the nodes recursively form a sub-tree. Edges are computed only as needed within each sector, and inter-sector edges are skipped. The algorithm takes $\mathcal{O}(n \log n)$ time, which is efficient enough even if n goes up to a few million. We call the result structure the LAMDA tree.

A. Algorithm Description

The LAMDA algorithm borrows the idea of divide-and-conquer. With the input node set V, the algorithm first sets up a polar coordinate system, with a given root r as the pole. The polar axis is the ray pointing from r to north. Then, any node u can be located by (l, θ), where l is the geographic distance between u and r, and θ is the clockwise angle from the polar axis to u. Fig. 1 shows an illustrative example. According to the ascending order of θ, the *divide* step partitions the Euclidean space into multiple sectors. There are two conditions to create a new sector: partition angle, and partition size. The partition angle α, as a configurable parameter, is the maximum sector angle, while the partition size $\frac{d^h-1}{d-1}$ is the maximum number of nodes in a sector, which is the number of nodes in a full h-level d-ary tree. (Please refer to Section IV-B for detailed discussions regarding α, d, and h.) Breaking either condition will cause the algorithm to seal the current sector and start a new one. Define the sector root as the node with the minimum l in the sector. When a sector is sealed, a sub-LAMDA tree will be constructed in the sector, with $h-1$ as the hop height bound, and the sector root as the input root node for the recursive call. The *conquer* step connects r to all sector roots.

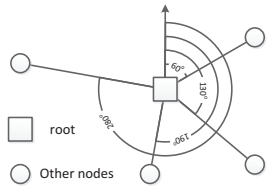


Fig. 1. Polar Angles

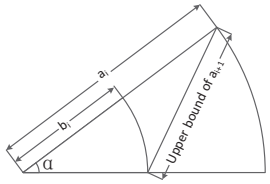


Fig. 2. Recurrence Formula

Algorithm details are shown in Algorithm 1. It takes 5 inputs: the node array V, the root node r, the degree constraint d, the hop height constraint h, and the partition angle α. Line $2 \sim 5$ checks if the recursive call arrives at a parent of leaf nodes. A positive answer means that r only has direct children without further descendants. Therefore, the algorithm appends all nodes in V to r's children array. Line $6 \sim 9$ sets up the polar coordinate system, and computes the coordinates of all nodes in V with given root r. Line 10 sorts all nodes based on the polar angle θ. Line $13 \sim 25$ scans all nodes according to the sorted order, and performs divide-and-conquer steps. Line 15 checks if the current state meets either partition condition. If not, the algorithm simply updates U on line 16. Otherwise, the algorithm starts a new recursive call on line 21 with U as the input node set, sector root rr as the input root node, and $h - 1$ as the hop height constraint.

Algorithm 1 LAMDA

Input: Node set V, Root node r, Degree constraint d, Tree height constraint h, Partition angle constraint α.
Output: A tree structure rooted at r.

1: **procedure** LAMDA(V, r, d, h, α)
2: **if** $h \leq 2$ **then** /* Reaching a parent of leaf nodes. */
3: $r.children \leftarrow V$
4: **return** r
5: **end if**
6: **for** v **in** V **do** /* Set up polar coordinate system. */
7: $v.\theta \leftarrow$ compute v's polar angle with given root r
8: $v.l \leftarrow$ compute the distance between v and r
9: **end for**
10: Sort V according to ascending order of θ
11: $minAngle \leftarrow 0$ /* the mini θ in the current sector*/
12: $U \leftarrow \emptyset$ /* Node set for the next level recursive calls*/
13: **for** $i \leftarrow 1$ **to** $|V|$ **do** /* Scan through all nodes to do partition.*/
14: $v \leftarrow V[i]$
15: **if** $v.\theta - minAngle \leq \alpha$ **and** $|V| \leq \frac{d^{h-1}-1}{d-1}$ **then**
16: $U \leftarrow U \bigcup \{v\}$
17: **else** /* Seal a sector, and start recursive call. */
18: $rr \leftarrow$ the node in U with smallest l
19: $U \leftarrow U \setminus \{rr\}$
20: $r.children \leftarrow \{rr\} \bigcup r.children$
21: LAMDA(U, rr, d, $h-1$, α)
22: $minAngle \leftarrow v.\theta$
23: $U \leftarrow \{v\}$
24: **end if**
25: **end for**
26: **return** r
27: **end procedure**

B. Algorithm Analysis

Complexity: Each recursive call takes three major operations: compute node coordinates in the polar coordinate system, sort nodes based on θ, and partition them according to the sorted order. Setting up the polar coordinate system, and partitioning the nodes both take $\mathcal{O}(n)$ time. If θ is stored with fixed precision, the sorting on line 10 also takes $\mathcal{O}(n)$ with linear sorting algorithms, such as radix sort [20]. Let $T(n)$ denote the computational complexity of LAMDA with n nodes. Then, the recurrence relation is:

$$T(n) = (d + \frac{2\pi}{\alpha} - 1)T(\frac{n}{d + \frac{2\pi}{\alpha} - 1}) + \mathcal{O}(n). \quad (1)$$

According to the Master Theorem ([20]), the complexity of LAMDA is:

$$T(n) = \theta(n^{\lg_{d+\frac{2\pi}{\alpha}-1}(d+\frac{2\pi}{\alpha}-1)} \lg^{0+1} n) = \theta(n \lg n). \quad (2)$$

Approximation Ratio: Besides the algorithm complexity, another very important property of the algorithm is the approximation ratio. To prove the approximation ratio, we need an estimation of the global optimal solution. As the LAMD tree connects all nodes, one trivial lower bound of the optimal solution is the largest l of all nodes with respect to the top level root r. Let L denote this lower bound. We use L to prove the approximation ratio of our algorithm.

Each recursive call has a different root r, and a new polar coordinate system is set up with respect to r. Denote a_i^j and b_i^j as the largest and smallest $v.l$ for all nodes v in the j^{th} sector of recursive level i, where l is the distance between v and r. Let $a_i = \max_j a_i^j$, and $b_i = \max_j b_i^j$. When constructing the LAMDA tree, the root always connects to the nearest node in its subtree. Hence, the physical meaning of b_i is the maximum edge length in the tree between level $i-1$ and level i. Clearly, the distance height of the tree is bounded by the length sum of longest edges of all level (*i.e.*, $\sum_{i=1}^{h} b_i$). Thus, the worst case approximation ratio is upper bounded by $\frac{\sum_{i=1}^{h} b_i}{L}$. Please note, except a_1 (equals to L), a_i and b_i depend on the algorithm input. With fixed input, both a_i and b_i are deterministic. In the analysis, we consider the worst case approximation ratio for any input. Hence, both a_i and b_i are considered as variables (except a_1).

As shown in Fig. 2, let us consider a sector with $\alpha < 60^o$, and derive the relationship between a_i, b_i, and a_{i+1}. According to Law of Cosines, the upper bound of a_{i+1} with respect to a_i and b_i is:

$$a_{i+1}^2 \leq a_i^2 + b_i^2 - 2a_i b_i \cos\alpha. \quad (3)$$

Hence, to find the worst possible approximation ratio, we need to solve the following problem:

$$\begin{aligned} \max \quad & \sum_{i=1}^{h} b_i, \\ s.t. \quad & a_{i+1}^2 \leq a_i^2 + b_i^2 - 2a_i b_i \cos\alpha, \\ & 0 \leq b_i \leq a_i, a_1 = L. \end{aligned} \quad (4)$$

Theorem 1. *With any input node set, and $\alpha < 60^o$, $\frac{\sum_{i=1}^{h} b_i}{L}$ induced by the* LAMDA *algorithm is bounded by $\frac{1}{1-\sqrt{2-2\cos\alpha}}$ for any tree hop height h.*

Proof: We prove Theorem 1 by applying induction on LAMDA Tree hop height h.

- **Basis:** For $h = 1$, $b_1 \leq a_1 = L \leq \frac{L}{1-\sqrt{2-2\cos\alpha}}$. Therefore, the proposition holds trivially.
- **Inductive Step:** We assume for $h = k$, $\sum_{i=1}^{k} b_i \leq \frac{L}{1-\sqrt{2-2\cos\alpha}}$. Then, for $h = k+1$, we have:

$$\sum_{i=1}^{k+1} b_i = b_1 + \sum_{i=2}^{k+1} b_i. \quad (5)$$

Let $b_i' = b_{i+1}$, $a_i' = a_{i+1}$, and $a_1' = L'$, for $i \geq 1$. Let β denote $\cos\alpha$, and let γ denote $1 - \sqrt{2-2\cos\alpha}$.

$$\sum_{i=1}^{k+1} b_i = b_1 + \sum_{j=1}^{k} b_j' \leq b_1 + \frac{L'}{\gamma} = b_1 + \frac{a_2}{\gamma}. \quad (6)$$

Then with inequality 3, we have:

$$\sum_{i=1}^{k+1} b_i \leq b_1 + \frac{\sqrt{a_1^2 + b_1^2 - 2a_1 b_1 \beta}}{\gamma}. \quad (7)$$

Let $G(b_1)$ denote the right side of Inequality 7. We then evaluate the maximum value of $G(b_1)$ by examining the derivative of $G(b_1)$.

$$\frac{\mathbf{d}G(b_1)}{\mathbf{d}b_1} = 1 + \frac{b_1 - \beta a_1}{\gamma\sqrt{a_1^2 + b_1^2 - 2\beta a_1 b_1}}. \quad (8)$$

The condition for $\frac{\mathbf{d}G(b_1)}{\mathbf{d}b_1} > 0$ is:

$$b_1 > a_1 \left(\frac{\beta\sqrt{1-\gamma^2} - \gamma\sqrt{1-\beta^2}}{\sqrt{1-\gamma^2}} \right). \quad (9)$$

Therefore, $G(b_1)$ first decreases and then increases, and the maximum value is achieved at boundary point 0 or a_1. By evaluating both boundary point, we conclude that the statement $\sum_{i=1}^{k+1} b_i \leq \frac{a_1}{\gamma}$ holds for both $b_1 = 0$, and $b_1 = a_1$. Hence, the approximation ratio is upper bounded by $\frac{L}{1-\sqrt{2-2\cos\alpha}}$.

■

In the LAMDA algorithm, d is a soft constraint, such that some node may have at most $d + \lceil \frac{2\pi}{\alpha} \rceil - 1$ children in worst case. It is because some partition may reach the partition size

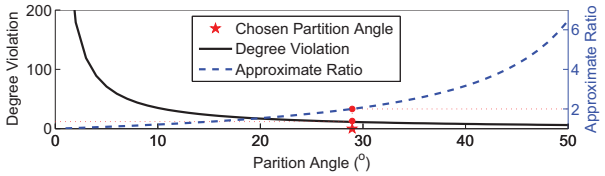


Fig. 3. Approximate Ratio vs Partition Angle

bound before reaching the partition angle bound, leaving the actual sector angle smaller than α. We call such sector a full sector. Within one recursive call, there are at most d full sectors, otherwise the number of nodes in sub-sectors will exceed $|V|$ in this call (*i.e.*, $d\left(\frac{d^{h-1}-1}{d-1}\right)+1 \geq \frac{d^h-1}{d-1}$). The 2π angle can be split into at most $\lceil\frac{2\pi}{\alpha}\rceil$ sectors with angle α. To create a worst case input that forces LAMDA algorithm to split the 2π angle with $\lceil\frac{2\pi}{\alpha}\rceil$ sectors, some nodes have to be spread out evenly in terms of θ, leaving at most $d-1$ full sectors. Putting them together, the children number of any node is bounded by $d+\lceil\frac{2\pi}{\alpha}\rceil-1$. Consider a worst case example, where $|V|$ is large enough. Let $|V|-2\lceil\frac{2\pi}{\alpha}\rceil$ nodes located on the ray with $\theta=0$. The polar angles (θ) of the other $2\lceil\frac{2\pi}{\alpha}\rceil$ nodes follow the sequence $(\epsilon, \alpha+\epsilon, \alpha+2\epsilon, 2\alpha+2\epsilon, ..., \lceil\frac{n-1}{2}\rceil\alpha+\lceil\frac{n}{2}\rceil\epsilon, ..., 2\pi-\epsilon)$.

However, as long as α is large enough, the extra overhead is negligible. For example, with $\alpha=29^o$, LAMDA tree achieves 2-approximation, and the worst case d violation is bounded by 12. For any modern computer, the cost of sending out 12 more short messages is negligible. Therefore, this violation does not affect the practicality of the algorithm. Fig. 3 shows the mapping between α, d violation, and approximation ratio.

V. RANDOMIZED LAMD TREE

The LAMDA algorithm achieves both low computational complexity and appealing approximation ratio. One may construct a LAMDA tree over the entire receiver set to disseminate the new message by setting the OSN server as the root. Any node that participates in the dissemination tree calls for only two pieces of data: the topology of the subtree rooted at itself ($topo$), and the new message (msg). These two data fields are encapsulated in the payload of an application level packet. On receiving this packet, the node only needs to forward the first level subtree of $topo$ and msg to the corresponding child.

When applying LAMDA in practice, we have three more concerns. First, the size of $topo$ linearly proportionally relates to the size of the subtree rooted at the receiver node. LAMDA tree, as a deterministic algorithm, always chooses the nearest node as the next hop. Hence, the nodes closer to the the data center are more likely to receive larger $topo$ payload than those far away nodes, leaving the load unbalanced on user side. Second, user nodes may fail (*e.g.*, crash or leave) during the message dissemination before finishing their jobs. If one user node fails, all of its descendants will not be able to receive the message. Therefore, some redundancies are needed to compensate the possible failures. Again, as LAMDA tree algorithm is deterministic, it always generates the same dissemination tree for the same input, making redundancies useless. Third, for a large receiver set, it may not be feasible to keep both d and h small at the same time, as $\frac{d^h-1}{d-1}$ (the number of nodes in a full h-level d-ary tree) has to be equal to or larger than $|V|$. To attack the aforementioned problems, we present Randomized LAMD Tree (LAMDR) algorithm which employs LAMDA tree algorithm as a subroutine. The generated tree structure is called LAMDR tree.

Randomized LAMD tree is a message dissemination tree rooted at the OSN data center. The motivating intuition is that, the data center is much more reliable and powerful than normal OSN user nodes. Hence, the algorithm asks the data center to send out more messages or carry out more computation when necessary. Compared to LAMDA algorithm, differences come from two aspects. First, the top-level root (data center) randomly chooses one user node from each sector as its direct child, rather than always picking the nearest one. The randomness only applies to the top level, as the approximation ratio deteriorates with the increase of randomness. According to our experiments shown in Section VII-C, this top-level randomness is enough to balance load among users. Second, the children number constraint d does not apply to the top-level root. In LAMDR algorithm, the top-level root first partitions V into sectors with at most $\frac{d^{h-1}-1}{d-1}$ nodes. Then a LAMDA tree is constructed within each sector. Thanks to the randomization in the top-level children selection, repeat LAMDR algorithm s times will generate s different dissemination trees, which allows redundancies to be applied.

Algorithm 2 LAMDR

Input: Node set V, root r, degree constraint d, tree height constraint h, partition angle constraint α, number of redundancy copies s.
Output: A set of s LAMDR trees R.

1: **procedure** MAIN(V, r, d, h, α, s)
2: $R \leftarrow \emptyset$ /* The set to hold the roots of all LAMDR trees.*/
3: **for** $i \leftarrow 1$ **to** s **do** /* repeat LAMDR s times. */
4: $R \leftarrow \{\text{LAMDR}(V, r, d, h, \alpha)\} \bigcup R$
5: **end for**
6: **return** R
7: **end procedure**

1: **procedure** LAMDR(V, r, d, h, α)
2: **for** v **in** V **do** /* Set up polar coordinate system. */
3: $v.\theta \leftarrow$ compute v's polar angle with given root r
4: $v.l \leftarrow$ compute the distance between v and r
5: **end for**
6: Sort V according to ascending order of θ
7: $U \leftarrow \emptyset$ /* node set for the next level subtree. */
8: **for** $i \leftarrow 1$ **to** $|V|$ **do**
9: $v \leftarrow V[i]$
10: **if** $|U| \leq \frac{d^{h-1}-1}{d-1}$ **then**
11: $U \leftarrow U \bigcup \{v\}$
12: **else** /* Seal the current sector and build a LAMDA. */
13: $rr \leftarrow$ randomly pick one node from U
14: $U \leftarrow U \setminus \{rr\}$
15: $r.children \leftarrow \{rr\} \bigcup r.children$
16: LAMDA(U, rr, d, $h-1$, α)
17: $U \leftarrow \{v\}$
18: **end if**
19: **end for**
20: **return** r
21: **end procedure**

The pseudo code of LAMDR is shown in Algorithm 2. Compared to LAMDA, LAMDR takes one more input parameter: the number of redundant copies, denoted by s. The MAIN procedure builds s LAMDR trees by calling the LAMDR procedure repeatedly. In the LAMDR procedure, line 2 $\sim$ 5

sets up the polar coordinate system with given root r as the pole. Line $8 \sim 19$ scan through all nodes in V according to the ascending order of θ to build the LAMDR tree. Line 10 checks if the current sector reaches the size bound. If no, it adds v into the current sector on line 11. Otherwise, the algorithm randomly chooses a node rr from U as direct child on line 15, and calls the LAMDA algorithm on Line 16 to construct a LAMDA tree with U as the input node set, rr as the root, and $h-1$ as the hop height bound.

LAMDR algorithm addresses all of the aforementioned three concerns. First, user node failures can be compensated by choosing the right value of s. Second, thanks to the randomization, any user node is equally possibly to be the direct child of the data center, and hence, the load spread out evenly. Third, LAMDR does not apply d constraint to the data center node. Therefore, d and h can be reasonably small at the same time.

VI. IMPLEMENTATION

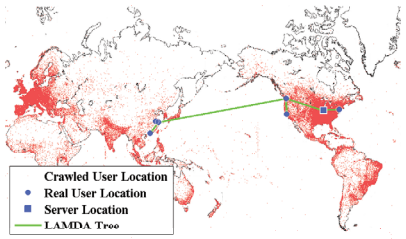


Fig. 4. Twitter User Location

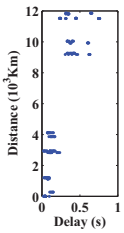


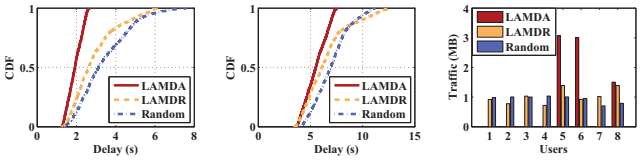
Fig. 5. Delay

To evaluate the performance of Centaur, we implement both large scale simulations with real Twitter and Google data and a web-based prototype by using Adobe Cirrus [21].

A. Crawlers

We develop a few crawlers to get user profile, follower set, and tweets information from Twitter. We first use the sample API [22] to collect more than 1 million tweets on October 22nd 2012. To speed up the user graph crawling phase, we utilize 20 machines with different IP addresses. After more than 3 months, we have crawled roughly 1.2 billion users and 1.5 billion edges.

The next step is to gather user locations. Twitter can easily get that information internally, as servers know users' current IP addresses. Unfortunately, this part of the information is not publicly available. We try another way around. From the data given by "users/lookup" API [22], we found that many users specify a short string in the "location" field to indicate their locations. We submit this string to Google Geocoding API [23] to translate it into latitude-longitude coordinates. Google has rate-limited each IP address to submit at most 2500 request per day. As many users share the same location string especially for people in big cities (*e.g.*, many people just write "New York" as their location), we first extract the distinct location string set, which contains more than 2 million strings. With months of parallelized crawling, we managed to translate those "distinct" location strings into more than 300 thousand "distinct" coordinations. The number of "distinct" coordinations is much smaller than the number of "distinct" location strings, because many different



(a) Average Delay (b) Maximal Delay (c) Outgoing Traffic

Fig. 6. Real-world Experiment Results

strings are representing the same coordination (*e.g.*, "New York City", and "the Big Apple"). Fig. 4 shows the location crawling result. Each small dot corresponds to one distinct user location.

B. Web Based Centaur

The ultimate goal of Centaur is to implement all the user side logics in web pages, rather than require extra user efforts to download dedicated softwares to do that. We have implemented a prototype system based on Adobe Cirrus [21]. Cirrus is web based P2P framework. Upon open, the Centaur web page will get a peer ID from Cirrus, and translate the current IP address into coordinations using IPInfoDB [24] and Google Geocoding [23]. Then, the Centaur web page sends those information to the Centaur server. On closing, the web page also sends a notification to the Centaur server. In this way, the server keeps track of the online user set. When a new message is published, the Centaur server will compute three LAMDR tree, and disseminate the message according to the tree structure. The evaluation involves 8 users in 7 different cities, namely Champaign (2 users), Mountain View, Nanjing, New York, Seattle, Shanghai, and Shenzhen. The 7 cities are marked with rounds in Fig. 4. Fig. 5 illustrates how geographical distance relates to the RTT time of pushing 64 bytes data between two users. The clear increase trend supports our assumptions.

VII. EVALUATION

We compare Centaur to two other different solutions: Random Tree, and Cuckoo [3]. We design Random Tree algorithm as a base line solution, while Cuckoo is the state-of-the-art distributed microblogging system.

Random Tree also utilizes user assisted message dissemination. However, it is oblivious of the user location information. Upon receiving a new message, the server randomly shuffles the online follower set, and then, cut the set into blocks with at most $(d^h-1)/(d-1)$ followers in each block. After that, each block (only contains node information without edge information) along with the new message is sent to a random user in that block, and the receiving follower will further propagate the message in the similar way. Hence, the followers actually form a dissemination tree with randomized topology. Random Tree also employs the same level redundancies as Centaur to compensate node failures.

Cuckoo is a P2P microblogging system, that mostly relies on individual users' storage, computation, and network resources. The publisher of a message is responsible for either notifying all followers, or initializing gossip-based message propagations, depending on the size of the follower set. If the follower set is small, the publisher's machine will directly notify all followers by itself. Otherwise, the publisher will send the new update to $\mathcal{O}(\log n)$ followers, where n is the

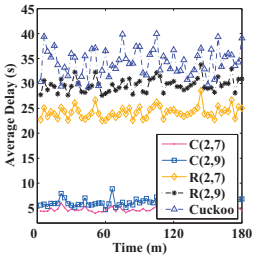


Fig. 7. Average Delay

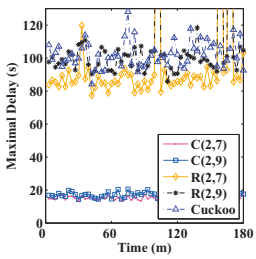


Fig. 8. Maximal Delay

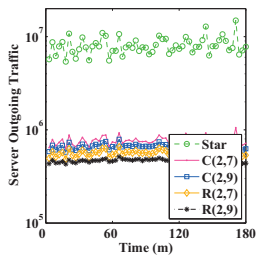


Fig. 9. Server Out Traffic

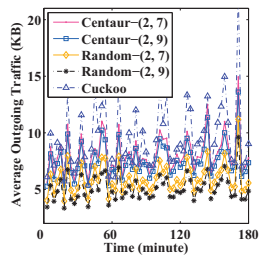


Fig. 10. User Out Traffic

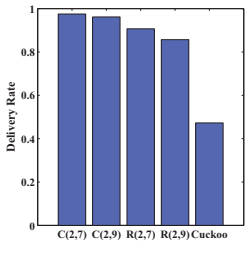


Fig. 11. Delivery Ratio

size of the follower set. When the follower receives the first notification, he will forward the message to other $\mathcal{O}(\log n)$ followers.

A. Experiment Setup

We carry out both small-scale real-world evaluations and large-scale simulations.

Real-world Evaluation: In small-scale real-world evaluations, 8 users in 7 different cities stay online during a 2-hour time period. As the number of users is small, we do not allow them to disconnect during the experiment, and all users form a complete graph (*i.e.*, each user is a follower of all other users). The evaluation compares LAMDA, LAMDR, and Random Tree algorithms. Both LAMDA and LAMDR are bounded by $d = 2$ and $h = 3$. The LAMDA tree structure is mark on Fig. 4.

Large-scale Simulation: In large scale simulations, we further evaluate different settings with each solution. Let *C-(d ,h)* and *R-(d,h)* denote the experiment using Centaur algorithm or Random Tree algorithm with children number bound d and tree hop height bound h. The parameter α is chosen as 29^o in all experiments.

During packet level simulations, we replay the 4 hour tweet trace collected by using Twitter sample API [22] from 8PM to 12PM on October 22nd 2012. Users join and leave dynamically during the simulation. When join, the user chooses one from the crawled user locations as shown in Fig. 4. Every second, each user will leave with probability of 1%. When one user leaves, another different user will be created at the same time. In this way, we maintain 150,000 dynamic online users. Both user overall incoming and overall outgoing bandwidth are limited by 20 KB. If the required bandwidth goes beyond the limit, packets will be queued at the sender side. For each new tweet, we assume that 1% of the publisher's followers are online.

There are three types of delays in the simulation: propagation delay, transmission delay, and queueing delay. The propagation delay linearly relates to geographic distances. According to literatures [16, 17], the delay increases by 1MS every 50KM. To simulate the transmission delay, we employ the TCP flow control protocol. Every message ($topo$ + msg) is cut into 1KB packets. As most messages are short (just a few KB), the flow finishes before the end of TCP slow start. The transmission delay also positively relates to geographic distances [25, 26]. As each user updates sporadically, the queueing delay is expected to be small.

B. Delivery Delay

We first evaluate the message delivery delay. In the evaluation, each parent node records the time difference between a message is sent to and its ACK is received from the child, which we call link delay. In order to avoid introducing errors in synchronizing clocks of different users, the delivery delay of each message in the evaluation is measured by the sum of link delays along the path from each user to the server. Fig. 6 (a) and (b) show the CDF of average and maximal message delivery delay respectively. LAMDA is doing much better than the other two, and LAMDR does not considerably outperform Random Tree algorithm. The reason is that, LAMDR assigns children of the server randomly to achieve better load balancing. In such a small-scale experiment, the tree built by LAMDR is at most of height 3. Therefore, even one level of randomness will introduce significant relative performance degradations.

As simulations are executed on a single machine, we employ a global clock for the server and all users. Each application packet carries a timestamp of when it is sent by the server, and the delivery delay is calculated on the receiver side by comparing the current time with the timestamp. Time is divided into 3-minute blocks. The average delay averages the delivery delay inside each time slot, while the maximal delay is the largest delay measured in the time slot. Fig. 7 and Fig. 8 show simulation results. Clearly Centaur achieves the shortest delay. The delivery delay induced by Random Tree algorithm is roughly 6 times longer with respect to the average delay, and roughly 4 times longer with respect to the maximal delay. It is because, the Random Tree algorithm does not take the geographic information into account, and hence each overlay hop in the dissemination tree can be much longer than in the Centaur case. For example, we have three online nodes: server in New York, Bob in Shanghai, and Cathy in D.C, and they are expected to form a path to disseminate message from the server to the other two. With Random Tree, it is possible that the path will be server $\rightarrow$ Bob $\rightarrow$ Cathy. Hence, the message go across the Pacific ocean twice, while server $\rightarrow$ Cathy $\rightarrow$ Bob is obviously a better solution.

Fig. 7 and 8 also show that with the increase of h, the delay also increases to some degree. The first reason is that larger tree height will result in higher failure rate. Therefore, for one user node, the fastest notifying message copy is likely to get dropped, forcing the user to wait for slower copies. Besides, the packets may get queued at the user nodes, and the queueing chance increases with the increase of hop number.

Cuckoo takes even longer time to deliver messages when compared to Random Tree. It is because, 1) Cuckoo does not take geographic information into account, 2) it does not even have a centralized server to help calculate a well-structured tree. The resulting gossip-based message dissemination topol-

ogy is ad hoc, and some paths can be excessively long with respect to both hop number and geographic distances.

C. User and Server Traffic

For practicality reasons, Centaur should not add too much overhead to individual users. In Centaur, each user only needs to forward an application packet containing $topo$ and msg to each of his or her children in the received tree structure. As the user side logic is very simple, the computational overhead is not a concern. Therefore, we focus on the communication overhead. Fig. 6 (c) shows the accumulated outgoing network traffic of 8 users. Although LAMDA achieves good performance in terms of delivery delay, its communication workload is highly skewed. LAMDR algorithm clearly balances Centaur's load into a much more uniform distribution. Random Tree algorithm balances users' load most evenly as expected. Fig. 10 plots average user outgoing traffic in the simulation, which further confirms that Centaur introduces negligible communication overhead to users.

To quantify the savings of servers' outgoing traffic, we introduce another baseline solution to simulations which is called the Star Tree.

Star Tree solution resembles to the current OSN message dissemination implementations the most. The server is responsible for delivering every single message, and users are just simple-logic receivers. In simulations, the server aggressively pushes new messages to followers, rather than handling user polls passively. Besides, we assume the server has unlimited outgoing bandwidth, such that the message will not get queued on the server side.

As we are utilizing a large number of users to perform message dissemination, the server side load is expected to be greatly reduced. As shown in Fig. 9, Centaur does achieve that. The saving given by Centaur is more than one order of magnitude compared to the Star Tree solution. Random Tree wins a little bit with respect to server outgoing traffic. The reason is that the server does not need to send out any edge information, as it is randomly computed at the user side. Cuckoo is not applicable here, as the Cuckoo server does not actively participate in the message dissemination phase.

D. Delivery Rate

The delivery rate is also a crucial metric. One solution is useful and reasonable only if the delivery rate is high enough. Fig. 11 shows the delivery ratio of different solutions. In the experiment, Both Centaur and Random tree uses redundancy 3. C-(2, 7) achieves about 98% delivery rate, while R-(2, 7) achieves 90%. We believe Random tree loses due to its long delivery delay. Since, every second, one user is possible to fail with probability 1%, longer delay means higher failure rate in the dissemination tree. In C-(2, 9) and R-(2, 9) experiments, the delivery rate decreases a little bit, because taller tree leads to longer delay and higher failure rate.

VIII. CONCLUSION

This paper elaborates the design and evaluation details of Centaur, a user-assisted message dissemination solution for Online Social Networks. The huge number of OSN publishers/channels, and the sporadic nature of each channel make OSN message dissemination distinct from earlier multicast, publish/subscribe, or P2P streaming scenarios. With Centaur, the server computes an well-shaped dissemination tree for every incoming message to connect all online followers of the message's publisher. In order to make this design practical, we design the LAMDR Tree algorithm which achieves $\left(\frac{1}{1-\sqrt{2-2\cos\alpha}}\right)$-approximation with $\mathcal{O}(n \log n)$ computational complexity. Since any user node in the dissemination tree may fail before finishing its job, leading all of its descendants disconnected from the tree, Centaur utilizes redundancies to compensate failures. The optimal redundancy level is computed to satisfy the given message delivery rate. Evaluation results show that Centaur saves more than an order of magnitude outgoing bandwidth on the server side compared to the star topology, and achieves very high message delivery rate with very low delay.

REFERENCES

[1] T. Sakaki, M. Okazaki, and Y. Matsuo, "Earthquake shakes twitter users: real-time event detection by social sensors," in *ACM WWW*, 2010.

[2] P. Earle, D. Bowden, and M. Guy, "Twitter earthquake detection: earthquake monitoring in a social world," in *Annals of Geophysics*, 2011, pp. 851–860.

[3] T. Xu, Y. Chen, L. Jiao, B. Y. Zhao, P. Hui, and X. Fu, "Scaling microblogging services with divergent traffic demands," in *Middleware*, 2011.

[4] "Facebook, twitter, google+, pinterest: The users of social media," http://www.mediabistro.com/alltwitter/social-media-users_b22556, November 2012.

[5] "Twitter turns six," http://blog.twitter.com/2012/03/twitter-turns-six.html, November 2012.

[6] B. Zhang, S. Jamin, and L. Zhang, "Host multicast: A framework for delivering multicast to end users," in *IEEE INFOCOM*, 2002.

[7] E. Brosh, A. Levin, and Y. Shavitt, "Approximation and heuristic algorithms for minimum-delay application-layer multicast trees," *IEEE/ACM Trans. Netw.*, pp. 473–484.

[8] Z. Li, T. Zhang, Y. Huang, Z.-L. Zhang, and Y. Dai, "Maximizing the bandwidth multiplier effect for hybrid cloud-p2p content distribution," in *IWQoS*, 2012.

[9] V. Ramasubramanian, R. Peterson, and E. G. Sirer, "Corona: A high performance publish-subscribe system for the world wide web," in *NSDI*, 2006.

[10] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Design and evaluation of a wide-area event notification service," *ACM Trans. Comput. Syst.*, vol. 19, no. 3, pp. 332–383, Aug. 2001.

[11] R. S. Kazemzadeh and H.-A. Jacobsen, "Publiy+: A peer-assisted publish/subscribe service for timely dissemination of bulk content," Macau, China, 2012, pp. 345–354.

[12] Y. Liu, L. Xiao, X. Liu, L. M. Ni, and X. Zhang, "Location awareness in unstructured peer-to-peer systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 2, pp. 163–174, Feb. 2005.

[13] T. Qiu, G. Chen, M. Ye, E. Chan, and B. Y. Zhao, "Towards location-aware topology in both unstructured and structured p2p systems," in *IEEE ICPP*, 2007.

[14] R. Bindal, P. Cao, W. Chan, J. Medved, G. Suwala, T. Bates, and A. Zhang, "Improving traffic locality in bittorrent via biased neighbor selection," in *IEEE ICDCS*, 2006.

[15] D. Niu, Z. Liu, B. Li, and S. Zhao, "Demand forecast and performance prediction in peer-assisted on-demand streaming systems," in *IEEE INFOCOM*, 2011.

[16] E. Katz-Bassett, J. P. John, A. Krishnamurthy, D. Wetherall, T. Anderson, and Y. Chawathe, "Towards ip geolocation using delay and topology measurements," in *ACM IMC*, 2006.

[17] B. Gueye, A. Ziviani, M. Crovella, and S. Fdida, "Constraint-based geolocation of internet hosts," *IEEE/ACM Trans. Netw.*, vol. 14, no. 6, pp. 1219–1232, Dec. 2006.

[18] "Neustar," http://www.neustar.biz, November 2012.

[19] Fitzsimons and Dennis, "Rhumb Lines and Map Wars: A Social History of the Mercator Projection by Mark Monmonier," *The Professional Geographer*, vol. 58, no. 4, pp. 497–499, Nov. 2006.

[20] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*, 2nd ed. McGraw-Hill Higher Education, 2001.

[21] "Adobe cirrus," http://labs.adobe.com/technologies/cirrus, November 2012.

[22] "Twitter rest api," https://dev.twitter.com/docs/api, November 2012.

[23] "Google geocoding api," https://developers.google.com/maps/documentation/geocoding/, November 2012.

[24] "Ipinfodb," http://ipinfodb.com, November 2012.

[25] I. F. Akyildiz, O. B. Akan, C. Chen, J. Fang, and W. Su, "Interplanetary internet: state-of-the-art and research challenges," *Comput. Netw.*, vol. 43, no. 2, pp. 75–112.

[26] M. Ghobadi, S. H. Yeganeh, and Y. Ganjali, "Rethinking end-to-end congestion control in software-defined networks," in *ACM Hotnets*, Seattle, WA, USA, 2012.