Social-Aware Replication in Geo-Diverse Online Systems

Stefano Traverso, Kévin Huguenin, Ionut Trestian, Vijay Erramilli, Nikolaos Laoutaris, Konstantina Papagiannaki

Social-Aware Replication in Geo-Diverse Online Systems

Stefano Traverso, Kévin Huguenin, *Member, IEEE,* Ionut Trestian, *Student Member, IEEE,* Vijay Erramilli, Nikolaos Laoutaris, and Konstantina Papagiannaki

Abstract—Distributing long-tail content is a difficult task due to the low amortization of bandwidth transfer costs as such content has limited number of views. Two recent trends are making this problem harder. First, the increasing popularity of user-generated content and online social networks create and reinforce such popularity distributions. Second, the recent trend of geo-replicating content across multiple points of presence spread around the world, done for improving quality of experience (QoE) for users.
In this paper, we analyze and explore the tradeoff involving the "freshness" of the information available to the users and WAN bandwidth costs, and we propose ways to reduce the latter through smart update propagation scheduling, by leveraging on the knowledge of the mapping between social relationships and geographic location, the timing regularities and time differences in end user activity. We first assess the potential of our approach by implementing a simple social-aware scheduling algorithm that operates under bandwidth budget constraints and by quantifying its benefits through a trace-driven analysis. We show that it can reduce WAN traffic by up to 55% compared to an immediate update of all replicas, with a minimal effect on information freshness and latency. Second, we build TailGate, a practical system that implements our social-aware scheduling approach, which distributes on the fly long-tail content across PoPs at reduced bandwidth costs by flattening the traffic. We evaluate TailGate by using traces from an OSN and show that it can decrease WAN bandwidth costs by as much as 80% and improve QoE. We deploy TailGate on PlanetLab and show that even in the case when imprecise social information is available, it can still decrease by a factor of 2 the latency for accessing long-tail YouTube videos.

Index Terms—Social networks, Content Distribution, Long-Tail, Geo-Replication

1 Introduction

Online content distribution technologies have witnessed many advancements over the last decade, from large content distribution networks (CDN) to peer-to-peer (P2P) technologies, but most of these technologies are inadequate in handling unpopular or long-tailed[1] content [1]. With the term long-tailed, we refer to web-objects (photos, videos, etc.), generally user-generated, which are requested a low number of times. Such contents represent a significant portion of the content found on OSNs. CDNs find it economically infeasible to deal with such content – the distribution costs for content that will be consumed by very few people is higher than the utility derived from delivering such content [2]. Unmanaged P2P systems suffer from peer/seeder shortage and meeting bandwidth and QoE constraints for such content.

The problem of delivering such content is further exacerbated by two recent trends: the increasing popularity of user-generated content (UGC), and the rise of online social networks (OSNs) with large user bases distributed across the entire planet [3], [4], as a distribution mechanism that has helped reinforce the long-tailed nature of content. For instance, Facebook hosts more images than all other popular photo hosting websites [5], and they now host and serve a large amount of videos [6]. Content created and shared on social networks is predominantly long-tailed with a limited interest group, in particular if one considers notions such as Dunbar's number [7]. The increasing adoption of smartphones, which are capable of taking pictures or shooting videos and uploading the content to the OSN, will further drive this trend.

In order to deliver content and handle a geo-diverse user-base [8], [9], most large distributed systems are relying on geo-diversification, with storage in the network [3], [4], [10]. One can push content to geo-diversified Points of Presence (PoP) closest to the user, thus limiting the parts of the network affected by a request and improving latency for the user. However, it has been shown that transferring content between such PoPs can be expensive due to WAN bandwidth costs [11], [12] and can raise consistency issues. For long-tailed content, the problem is more acute: one can *push* content to PoPs, only to have it not consumed, thus wasting bandwidth. Inversely one can resort to *pull* and transfer content only upon request, but this will lead to increased latencies and potentially contributing to the peak load. Given the factors above, along with the inability of current technologies to

- *Stefano Traverso is with the DET, Politecnico di Torino, 10129 Torino, Italy. E-mail: stefano.traverso@polito.it*
- *Kévin Huguenin is with the LCA1, School of Computer and Communication Systems, EPFL, 1015 Lausanne, Switzerland. E-mail: kevin.huguenin@epfl.ch*
- *Ionut Trestian is with the EECS Department, Northwestern University, 60208 Evanston, IL, USA.*
- *Vijay Erramilli, Nikolaos Laoutaris, and Konstantina Papagiannaki are with Telefónica I+D, 08019 Barcelona, Spain.*

1. The "long-tail" term was first coined by Chris Anderson in [1].

distribute long-tailed content [2], [13] while keeping bandwidth costs low, it appears that the problem of distributing long-tailed content is and will be a difficult endeavor.

The peak-based pricing policy used on WAN links raises a trade-off between the *freshness*[2] of the information available to the user and bandwidth costs. Specifically, the problem of finding a balance turns out to be a scheduling problem. Instantaneously transmitting content uploaded by users to replicas provides the best achievable information freshness (i.e., contents produced by a user at one PoP are made available immediately to the users at the other PoPs), however, as users' activity follows diurnal patterns with a peak at the end of the day [14], [15], the bandwidth costs are high for the total volume of data sent. Delaying updates to replicas reduces traffic peaks but comes at the price of reduced freshness (i.e., *staleness*).

Our Contribution: We first analyze the feasibility of our social-aware approach to scheduling with a simple algorithm that optimizes information freshness under WAN budget constraints by scheduling the transmission of updates, leveraging on specific aspects of geo-diverse OSNs. For example, our algorithm prioritizes the update of replicas that serve many of a user's friends, especially if these friends have a high probability of accessing the content. We couple this simple algorithm with a budget allocation mechanism that determines the bandwidth spent on updating each PoP.

Second, we present a practical system called TailGate, specifically designed to work online, and inspired by the above algorithm, that can distribute long-tailed content while lowering bandwidth costs and improving QoE. For the sake of practicality, TailGate does not operate under hard budget constraints but instead it flattens the traffic by distributing update replication over sliding time windows. The key to distribution is knowing (i) *where* the content will likely be consumed, and (ii) *when*. If we know the answers, we can *push* content wherever it is needed, at a time before it is needed, and such that bandwidth costs are minimized under peak based pricing schemes, e.g., 95th percentile pricing. In short, 95th percentile billing works as follows: The client, i.e., the social network operator using the link, is charged over a billing period (typically a month). The billing period is partitioned in intervals (typically 30-second to 5-minute intervals). During each intervals, the average bandwidth is computed. The 95th percentile is the value x such that 95% of the intervals have an average bandwidth lower than x. More details can be found in [16], [17]. Although in this paper we will focus on this pricing scheme, it needs to be stressed that lowering the peak is beneficial also under flat rate schemes or even with owned links, as in both cases network dimensioning depends on the peak bandwidth consumption.

More specifically, TailGate relies on rich and ubiquitous information: friendship links, regularity of activity and information dissemination via the social network. TailGate is built around the following notions that dictate the consumption patterns of users. First, users follow strong diurnal trends when accessing data [18]. Second, in a geo-diverse system, there exist time-zone differences between sites. Third, the social graph provides information on who will likely consume the content. At the center of TailGate is a scheduling mechanism that uses these notions. TailGate schedules content by exploiting time-zone differences that exist so as to spread and *flatten* out the traffic caused by moving content. The scheduling scheme enforces an informed *push* scheme, reduces peaks, hence the costs. In addition, the content is pushed to the relevant sites before it is likely to be accessed, thus reducing the latency for the end-users. Note that latency is indeed a critical issue as a small increase in latency can translate into a significant loss of revenues for the service providers [19].

The practical system proposed and the simple algorithm used for the motivation target the same problem of optimizing bandwidth usage when replicating content across a geo-diverse social network. However, they differ because the social-aware scheduling algorithm we present strictly flattens the traffic between PoPs (using hard limits) and schedules updates so as to minimize the staleness of content. The heuristic TailGate is instead an online solution that pushes content that is likely to be accessed at remote data centers. In this latter case, traffic is not flat as when we access content that is not available at the current PoP, we retrieve it from a remote one.

Results at a glance: In order to first understand characteristics of users that can be used by TailGate and if these characteristics are useful, we turn to a large dataset collected from an OSN (Twitter), consisting of over 8M users and over 100M content links shared. This data helps us understand where requests can come from, as well as give us an idea of when.

With regard to results, we first indicate using trace-driven simulations that social-aware scheduling algorithm can reduce WAN traffic by 55% without sacrificing information freshness. Also, for a given WAN bandwidth budget, our solution increases information freshness (wrt our penalty-based metric) by an order of magnitude compared to FIFO update scheduling.

As for the online solution, TailGate, we compare its performance in terms of reduction in bandwidth costs and improvement of latency for the user under different scenarios by using real data. When compared with a naive *push* policy, we see a reduction of the

2. We use the term information freshness here to denote the fact that, at a given instant, the content uploaded by a user is available to her friends, and it is captured by the penalty metric introduced in Section 3. We call *staleness* the fact that the information available to the user is not fresh (i.e., not up-to-date).

95th bandwidth percentile of 80% in some scenarios and a reduction of $\sim$ 30% over a *pull*-based solution that is employed by most CDNs. For only long-tailed[3] content, the improvement is even higher. We vary the quality of information available to TailGate and find that, even when it has less precise information, it still performs better than push and is similar to pull in terms of bandwidth costs, while lowering latency for up to 10 times of the requests over pull. We show that even in an extreme setting, where it has lower access to information, TailGate reduces latency by a factor of 2 when the users access long-tailed content.

The rest of this paper is organized as follows. In Sec. 2 we introduce the architecture of our system that we will use throughout the paper. In Sec. 3 we introduce the main contribution of this paper, the social aware scheduling algorithm, and the online solution inspired to it, named TailGate. Using the dataset described in Sec. 4 and in the supplementary material, we show the benefits brought by our social aware scheduling algorithm in Sec. 5 and we evaluate TailGate performance in Sec. 7 and in the supplemental material. We introduce several deployment scenarios based on who operates the system and what social information is available in Sec. 8, survey related work in Sec. 9, and conclude the paper in Sec. 10.

2 Preliminaries

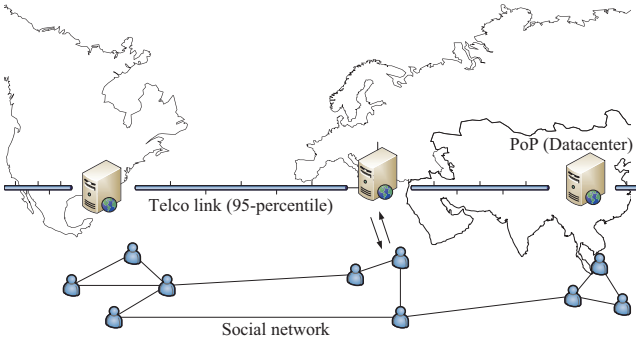

Fig. 1. Generic architecture: PoPs geo-distributed, handling content for geographically close users. Content created is first uploaded to the closest PoP, then this content is distributed to other PoPs.

For the sake of exposition, we describe a generic distributed architecture that will provide the template for the design and analysis of our solutions. In Sec. 8, we show how this architecture can be used for different scenarios: OSN providers and CDNs.

2.1 Architecture

We consider an online service with users distributed across the world. In order to cater to these users, the service is operated on a geo-diverse system comprising multiple points-of-presence (PoPs) distributed globally. These PoPs can be data centers managed by an OSN provider such as Facebook or CDN nodes. These PoPs are connected to each other by links. These links can be owned by the entity that owns the PoPs (for instance, online service providers such as Facebook or a Telco-operated CDN) or the bandwidth on these links can be leased from network providers.

Consider an application with a user base around the world. Users are assigned to and served out of their nearest (geographically) PoP, for all their requests. Placing data close to the users is a maxim followed by most CDNs, replicated services, as well as research proposals such as Volley [20]. Therefore all content uploaded by users is first uploaded to the nearest respective PoP. When content is requested by users, the nearest PoP is contacted and, if the content is available there, the request is served. The content can be present at the same PoP if content was first uploaded there or was brought there by some other request. If the content is not available, then a *pull* request is made and the content is brought to the PoP and served. This is the de facto mechanism (also known as a cold-miss) used by most CDNs [21]. We use this 'serve-if-available', or '*pull*-when-not available' mechanism in some cases, as the baseline and we show that this scheme can lead to high bandwidth costs. An example of this architecture is shown in Fig. 1 where there are multiple interconnected PoPs around the world each serving a local user group.

2.2 Why Social Aware Scheduling is Necessary?

Consider user Bob living in Boston and assigned to the Boston PoP. Bob likes to generate and share content (videos, photos) with his friends and family. Most of Bob's social contacts are geographically close to him, but he has a few friends on the West Coast, in Europe and in Asia. These geographically distributed friends are assigned to the nearest PoP, respectively. Bob logs in to the OSN at 6PM local time (peak time) and uploads a family video he wants to share. [4]

Like Bob, many users perform similar operations. A naive way to ensure this content to be as close as possible to all users before any accesses happen would be to *push* the updates/content to other PoPs immediately, at 6PM. Aggregated over all users, this process of pushing immediately can lead to a traffic spike in the upload link. Worse still, this content might not be consumed at all, thus having contributed to the

3. In the rest of the paper, we call long-tailed contents those that are requested less than 1,100 times. The threshold was determined experimentally: we observed a modal shift around this value in the popularity distribution of the contents in our dataset.

4. Note that the content uploaded might not be original and thus may already be hosted at some of the PoPs. In particular, such a situation occurs when a user shares a content posted by another user. In this case, the content is not replicated on the PoPs that already hold a copy of it. Such mechanisms, referred to as de-duplication, are widely used in cloud storage systems and are often implemented by means of hash functions. In the remainder of the paper, we assume that all the uploaded content is original.

spike unnecessarily. Alternatively, instead of pushing data immediately, we can wait till the first friend of Bob in each PoP accesses the content. For instance Alice, a friend of Bob's in London logs in at 12PM local time and requests the content, and the system triggers a *pull* request, pulling it from Boston. However, user activity follows strong diurnal trends with peaks (12PM London local), hence multiple requests by different users will lead to multiple pulls, leading to yet another traffic spike. The ineffectiveness of caching long-tailed content is well documented [2], and this problem is further exacerbated when Alice is the only friend of Bob's in London interested in that content and there are many such Alices in different geographic regions. Hence all these "Alices" will experience a low QoE (as they have to wait for the content to be downloaded) and the provider experiences higher bandwidth costs.

Our Approach: Instead of pushing content as soon as Bob uploads it, we wait till 2AM Boston local time, which is off-peak for the uplink, to push the content to London where it will be 7AM local time, again off-peak for downlink in London. Pushing the content at 7AM is still early enough to provide fresh content to Alice when she logs in, as she is not likely to do so before 12PM. Therefore Alice can access Bob's fresh content quickly, hence experience high QoE despite the fact that some delay was introduced. The provider has transferred the content during off-peak hours, thus decreasing costs. Our systems are built upon this intuition where such time differences, between content being *uploaded* and content being *accessed*, are exploited. In a geo-diverse system, such time differences exist anyway. In order to exploit these differences, we need information about the social graph (Alice is a friend of Bob), where a user's friend reside (Alice lives in London) and the access patterns of the friend (Alice will likely access it at 12PM).

2.3 System Requirements

Our systems need to address and balance the following requirements:

Reduce bandwidth costs: Despite the dropping price of leased WAN bandwidth, the growth rate of UGC, combined with the incorporation of media rich long-tailed content (e.g., images and videos), makes WAN traffic costs a major concern [12], [22].

Decrease latency: The latency is due to two factors: one is the latency in the link between the user to the nearest PoP; the other lies in getting that content from the source PoP, if not available at the nearest PoP. As the former is beyond our reach, we focus on placing the content to the closest PoPs.

Online and reactive: The scale of UGC systems [23] can lead to thousands of transactions per second, as well as a large volume of content being uploaded per second. In order to handle such volume, any solution has to be online, simple and react quickly.

2.4 What Social-Aware Scheduling Does Not Do

Social-aware scheduling optimizes for bandwidth costs and does not consider storage constraints. It is interesting to consider storage as well, but we believe the relatively lower costs of storage puts the emphasis on reducing bandwidth costs [21]. The system we envision in this paper is not dealing with dynamic content such as profile information etc. in OSNs as other systems do, e.g., SPAR [24] (note that SPAR reduces bandwidth costs by reducing the inter-PoP traffic volume but does not optimize the scheduling, unlike TailGate that reduces the costs by flattening[5] the inter-PoP traffic). The scheduling algorithm we propose and its online adaptation TailGate deal with large static UGC that is long-tailed and hence not amenable to existing solutions. Note that TailGate could still benefit from more sophisticated placement algorithms (instead of simply placing a user's profile on the closest PoP), such as SPAR..

3 Social-Aware Scheduling

We now introduce our social-aware scheduling algorithm for OSN to assess the potential of our approach. We then explain why such a solution cannot work online and introduce its practical adaptation, TailGate.

The problem addressed by our algorithm is in minimizing the perceived information staleness under cost constraints. It operates in two steps. First, it solves the problem of scheduling updates under a budget allocation among PoPs. Second, it uses a greedy heuristic to optimize the budget allocation. We compare the performance of the social-aware scheduler to the solution used in practice. The sizes of contents are chosen according to the distributions observed in popular video and photo hosting systems (e.g., Fickr and YouTube), accessed from Twitter. The bandwidth prices, for the cities where PoPs are located, obtained from a large ISP for a DS-3 link at 155Mbps are: London $30, Chicago $29, Houston $30, LA $29, Boston $27, Tokyo $80, and Delhi $370. The prices are given per Mbps of the 95th percentile of the link usage.

3.1 Scheduling

As previously mentioned, perceived information staleness is caused by the reads that occur before the updates. Therefore, a solution that minimizes perceived information staleness (captured by the penalty, in our evaluation) is to prioritize updates that are more likely to be imminently read and thus might incur a penalty. In a nutshell, the penalty is the total number of missing updates (because the replication of some updates is postponed by the scheduler) upon a read. Assume Alice, who lives in Europe, makes

5. By "flattening" the traffic, we mean decreasing the maximum (or the 95th percentile) of the bandwidth usage while keeping the volume of data constant.

2 posts at 10AM and that TailGate schedules them for replication to the US at 10:30AM and 11:00AM respectively. If Bob, who lives in the US, reads Alice's profile at 10:15AM, at 10:45AM, and at 11:15AM the total penalty is incremented by 3 (2 posts are missing on the US replica when Bob reads Alice's profile at 10:15, 1 post is missing for the 10:45 read, and 0 post are missing for the 11:15 read). We consider a discrete-time model in which time is divided into intervals corresponding to those used for the computation of the 95th percentile of the bandwidth usage (e.g., 1 hour). The scheduler decides which updates to replicate to other PoPs, at each time interval, based on a priority metric. For a given time interval t, the social-aware scheduler assigns a priority to each update based on the expected penalty and its size:

$$\mathrm{pr}(t, s, n, k) = \frac{1}{s} \sum_{m \in F_n \cap S_k} \rho_m^{[t]} \quad (1)$$

for an update of size s posted by user n and to be replicated on the k-th PoP. Here, F_n denotes the set of friends of user n, S_k denotes the set of users attached to the k-th PoP and $\rho_m^{[t]}$ denotes the probability that user m reads the profiles of its friends (e.g., its newsfeed) at time t. The priority is dynamically recomputed at every time t because the probability $\rho_m^{[t]}$ to read a profile varies over time. The updates are replicated by the social-aware scheduling algorithm by decreasing priority. The priority queue can be maintained by one specific PoP that organizes the replication between all PoPs. A decentralized solution is to make each PoP upload its local updates by decreasing priority. Similarly, each PoP downloads remote updates that originate from the other PoPs by decreasing priority. Note that due to the priority function in (1), selective replication is performed transparently: if a user has no followers attached to a given PoP, the priority of her updates to be replicated to this PoP falls to 0 and thus the updates are not replicated. In the case of 3 PoPs, selective replication reduces the total replication traffic volume by 55%.

3.2 Budget Allocation

As the number of users varies from one PoP to another, assigning evenly the budget between PoPs is sub-optimal. We thus optimize the budget allocation among PoPs by using a greedy algorithm run on a small learning period (say a day) and recomputed periodically to self-adapt to potential changes of user upload behaviors, new users joining, etc. The adjustment of the budget allocation should be performed at the beginning of a billing time period, over which the 95th percentile of the bandwidth usage is computed, as changing the budget within such a period cannot decrease the costs: If a given bandwidth cap has been used during the first half of the period, imposing a lower cap on the bandwidth for the rest of the period will not decrease the 95th percentile, hence the cost. Therefore, a typical time between two recomputations of the budget allocation is the billing period (typically 1 month) [17].

In a nutshell, the total budget is split in small budget increments (say \$1,000) that are iteratively assigned to one of the PoPs in such a way that the total penalty is minimized at each iteration. Consider the case of 3 PoPs. Initially, all PoPs are assigned a budget of \$0. At each iteration, we compute the total penalty when the budget increment is allocated to the first PoP, to the second, or to the third. For instance at the first step of the iteration, we run the scheduling algorithm for the following budget allocations: (\$1,000; \$0; \$0), (\$0; \$1,000; \$0) and (\$0; \$0; \$1,000). Assume that the corresponding penalties for the three budget allocations are 2000, 500 and 1000. Then the first budget increment is assigned to the second PoP as it brings the smallest penalty (i.e., 500 cold misses) and a new iteration is performed. The second iteration considers the following allocations: (\$1,000; \$1000; \$0), (\$0; \$2,000; \$0) and (\$0; \$1000; \$1,000). Assume that the corresponding penalties are 200, 450 and 300. Then the second budget increment is assigned to the first PoP (i.e., (\$1,000; \$1000; \$0)), and so on until the whole budget is allocated. (see the supplementary material for a pseudo-code version).

4 Datasets

As our solution aims at exploiting social information, the questions to consider are (i) what type of information is useful and available, (ii) how can such information be used? To answer these questions we used a large dataset obtained through a massive crawl of Twitter [25], which we completed by collecting location information and tweets. We ended up with 6.3M *active* users and approximately 499M tweets from which we extracted 101M URLs pointing to UGC. We considered several PoP locations and assigned users to the nearest PoP (w.r.t. the Haversine distance [26]). We focus our analysis on two time periods extracted from this trace. The first one we call **day** is the set of activities on the day with the maximum number of tweets in our dataset; and the second one we call **week** consists of the week showing traffic values closest to average computed over the dataset. For the read activity, we performed an analysis of the outgoing traffic on a link connecting a university in Italy to the Internet and we used SONG [27] to generate read patterns. One of our main findings is that on average 9.8% of the posted links were clicked on. More details about the datasets and its limitations for our intended usage model can be found in the supplementary material.

5 Social-Aware Scheduling

In this section we show how effective our social-aware scheduling policy can be in terms of bandwdith sav-

ings and in terms of paid penalties. First we will show how we assigned users to the closest PoP for QoS purposes. It is well known that contacts or 'friends' in social networks are located close together with respect to geographical distance. This fact can have large scale repercussions for assigning users to PoPs in an OSN. In order to understand the distribution of users and their contacts around the world, we proceed as follows. We consider 7 locations around the world (in this order): Boston, London, LA, Houston, Chicago, Tokyo, and Delhi, where we can place PoPs. We assign users to the nearest PoP.

Fig. 2 shows the link between geographical proximity and the number of followers. It shows the average number of followers assigned to the same PoP as the user, the average number of followers assigned to all other PoPs, and the maximum number of followers assigned to any other PoP. The considered PoPs are, in this order, Boston, London, LA, Chicago, Houston, Tokyo, and Delhi (see supplemental material). We see that the number of followers assigned to the same PoP is more than the maximum number of followers assigned to any other PoP, which suggests that users are assigned to PoPs where they have the most followers.

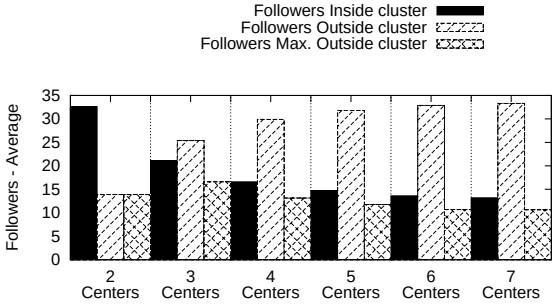

Fig. 2. Avg. number of followers inside/outside PoPs.

In order to get an idea of the distribution of followers across different PoPs, we compute the mean number of PoPs a user has followers in. For the 2-7 PoPs we consider, we obtain values from 1.2 to 2.6. In other words, on average a user has followers in 60% of the PoPs in the case with 2 PoPs; and has followers in 37% of the PoPs when we consider all 7 PoPs. This further suggests that, instead of copying content in 7 PoPs, the provider needs to copy content in only 2.6 PoPs. This result strongly suggests that exploiting the notion of *selective replication*, where users are replicated in PoPs where their followers are.

We compare our social-aware scheduler to FIFO scheduling where updates are replicated by decreasing age. For the sake of fairness, we complement FIFO with selective replication: updates are not replicated where the user has no followers. In Fig. 3, we plot the penalty as a function of the total monthly budget distributed among the PoPs using greedy allocation: each PoP receives the same bandwidth budget (represented on the x-axis). We vary the total budget from the budget needed to replicate all the updates over one time period and the budget needed to replicate all updates instantaneously. It can be seen that the social-aware scheduler outperforms FIFO by several orders of magnitude.

Encouraged by these promising results we developed the online adaptation of this algorithm, named TailGate which we present in the next section.

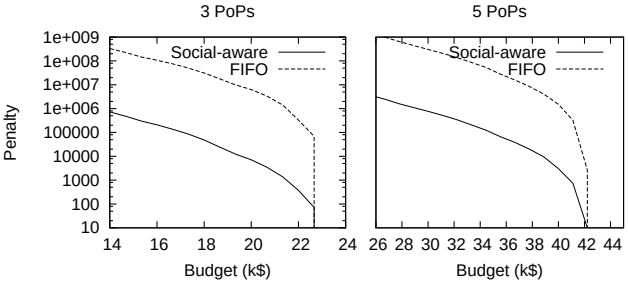

Fig. 3. Penalty function of the budget.

6 TailGate

In this section, we describe TailGate, a practical implementation of our social-aware scheduling approach.

6.1 Heuristic

To keep TailGate simple, we resort to a greedy heuristic for scheduling content. At a high level, we consider load on different links to be divided into discrete time bins (for instance 1 hour bins; this bins can correspond to those used for the 95th bandwidth percentile for pricing). Then, the heuristic is simple–given an upload performed by a given user at a given time at a given PoP that needs to be distributed to different sites–TailGate (1) determines (or estimates) the *first* bin in the future during which the content is likely be read by some uploader's friend, and then (2) schedule this content in the *least loaded* bin amongst the set of bins: [current bin, bin in which read occurs]. If more than one candidate bin is found, we pick a bin at random to schedule the content. Simultaneous uploads are handled randomly; no special preference is given to one upload over another. Consider the following example: Bob, who lives in Boston, uploads two videos at 10PM local time. Because Bob is friends with Alice, who lives in London, the videos must be replicated to the London PoP. Based on Alice's read patterns, Tailgate determines that Alice is most likely to read Bob's profile at 6AM local time (12AM in Boston). Given that the 10PM and 11PM time slots are the two least loaded, TailGate will schedule the replication of the first video in the 10PM time slot[6]. The second video is the scheduled for replication in the least loaded time slot, i.e., the 11PM slot.

6. The choice is done at random when 2 slots are equally loaded.

Assuming a static social network and steady read patterns for the users, TailGate needs to maintain, at each PoP, and for each tuple (user,time,destination PoP), the estimated time of the next read. Assuming that users are evenly distributed among the K PoPs, the number of users at each PoP is N/K. The number of time slots is proportional to the length T of a period of the read patterns (e.g., a day). In addition, each PoP maintains a replication queue to each of the other PoPs, and each queue contains (in the worst case) all the updates produced by its N/K users. Therefore, the data-structure used by TailGate grows linearly with N and T, and only the T factor is due to TailGate (FIFO/Push approach grows linearly with N only since it does not implement any scheduling). It can also be observed that the running time of TailGate grows linearly with N (to process each update of the queues) and with $\log T$ (to find the least loaded time slot among at most T and to update its load).

We highlight the salient points of this approach: (i) This is an online scheme in the sense that content is scheduled as it is uploaded. (ii) This scheme optimizes only for upload bandwidth; we tried a greedy variant where we optimized for upload and download bandwidth, but we did not see much improvement (results omitted for space reasons), so we settled for a simpler scheme. (iii) If we have perfect reads, TailGate produces no penalties by design. However, this is not the case and we quantify the tradeoff in the next section. (iv) In the presence of background traffic, one can use available bandwidth estimation tools to measure and forecast. (v) As the scheme relies on time difference between the current bin and the bin in which the content is likely to be read, the larger the difference, the better the results. (vi) Flash crowds are inherently unpredictable. TailGate performs as well as other schemes when there is a flash-crowd. However, as TailGate mainly deals with UGC (long-tailed), flash crowds will be rare, and in addition, UGC have very distinct (and slow) request characteristics [28].

6.2 Existing Solutions

We describe two solutions: Push/FIFO and a Pull-based approach that mimic various cache-based solutions (including CDNs), which can be used to distribute long-tailed content.

For all the schemes we consider, we assume that storage is inexpensive and that once content (e.g., a video) is delivered to a PoP, all future requests for that content originating from users of that PoP will be served locally. In other words, content is moved between sites only once. Flash-crowd effects are therefore handled by the nearest PoP. The key difference between the considered schemes is *when* the content is delivered.

Immediate Push/FIFO: The content is distributed to the PoPs as soon as it is uploaded. Assuming there is no loss in the network, FIFO decreases latency as content is always served from the nearest PoP.

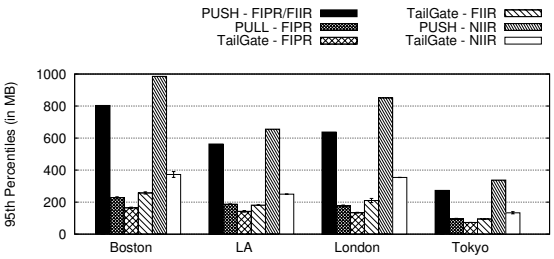

Fig. 4. 95th percentile bandwidth, for all contents.

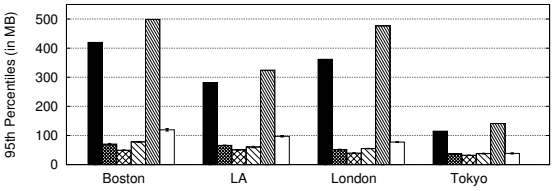

Fig. 5. 95th percentile bandwidth, for LT contents.

Pull: The content is distributed only when the first read request is made for that content. This scheme therefore depends on read patterns and we use the synthetic reads to figure out the first read for each upload. Note that in this scenario, the user who issues the first read will experience higher latency, as the object is fetched from a far away PoP across a WAN.

7 Evaluating TailGate

We now evaluate the practical adaptation of the social-aware algorithm presented in Section 3 TailGate by using the datasets of **day** and **week** described in Sec. 4 assuming the PoPs are located in Boston, London, LA and Tokyo. We evaluate TailGate against existing schemes, namely Push and Pull, under different models and we compare bandwidth costs and penalties.

7.1 Metrics

The main metric we use for comparison is the 95th percentile bandwidth of a traffic time-series for the given period under study. For the **day** and the **week** dataset, we calculate the 95th percentile bandwidth over 5 min bins. We also look at penalties, which is the number of requests pulled from the source and indicates higher latencies.

7.2 Scenarios Examined

We describe the scenarios we study that are designed to explore the effect of different types of information on performance metrics and to study where the benefits come from. The key inputs are the knowledge of reads and where these reads come from.

Full Information, Perfect Reads (FI, PR): In this model, we assume TailGate has access to the social

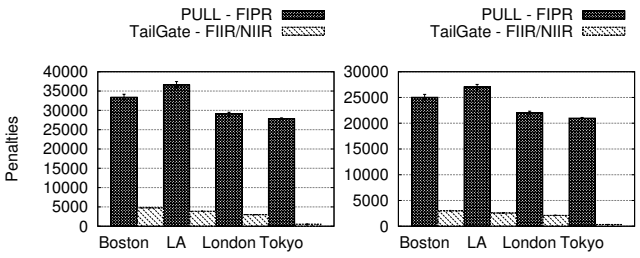

Fig. 6. Penalty at all PoPs for all (left)/LT (right) content.

graph. This helps in deciding where to distribute content: to only those PoPs that host friends of a user. In addition, we assume that TailGate has access to read patterns of users at a fine scale. Pull gets perfect reads as the transfer happens when a read request is made. In order to simulate the notion of perfect reads, we use reads that have been assigned to *individual* users (as described in Sec. 4). If we assume the probability of a friend of a user requesting the content is ρ, then at the k-th PoP the probability of requesting the content is $1-(1-\rho)^{|F_n \cap S_k|}$ where F_n is the set of user n's friends and S_k is the set of users attached to the k-th PoP. We use $\rho = 0.1$, based on the measure of the read activity shown in Sec. 4. In order to calculate penalties, we first note that for Push and TailGate there are no penalties, whereas for Pull, penalties is simply the number of uploads transmitted to different PoPs, as these uploads face higher latency.

Full Information, Imperfect Reads (FI, IR): In this model, we assume TailGate has access to the social graph, but has imperfect knowledge of read patterns; TailGate has access to generic read trends; diurnal patterns. In order to simulate the notion of imperfect reads, we use *generic* reads described in Sec. 4. Generic reads are based on a probability distribution that depends on the time of day (obtained by learning the diurnal patterns from the traffic observed in a recent past). In simulations, we draw samples from the aforementioned distribution to determine if a write was successful. Note that this is *conservative* and it stacks the odds against TailGate: we transfer more data close to the peak time, and, as the read probability is higher towards the peak, the set of bins that TailGate works with is lower. As there is inaccuracy in read information, TailGate can schedule data after the actual read happens – in effect leading to a Pull request to get data to serve that read request. Hence, TailGate suffers from penalties. In order to quantify this, we first extract the time when an upload is scheduled by TailGate under inaccurate information. Then we compare this against the schedule under accurate information. For all transmissions scheduled after the actual read request, there is a penalty.

No Information, Imperfect Reads (NI, IR): In this model, we assume TailGate has no access to the social graph and has imperfect knowledge of read patterns. In this case, content is distributed to all PoPs; bandwidth costs will consequently increase, as we pay the price of using limited/inaccurate read information.

7.3 Results

The results of our trace-driven simulations are presented in Fig. 4 and in Fig. 5. They report the 95th percentile of the bandwidth for all content (Fig. 4) and specifically for long-tailed (LT) content (Fig. 5). The results presented in Fig. 4 and Fig. 5 have to be observed together with those in Fig. 6, where we report the penalties paid by Pull and TailGate for all content (left plot) and for long-tailed content (right plot), respectively. We present results from our **week** dataset for space considerations, and all the results presented here assumes each PoP has one uplink and one downlink. We also studied the case where each PoP has seperate uplinks/downlinks to other PoPs (for instance service providers such as Google, Facebook, etc. and their geo-diverse data center system). The results for this and our `day` dataset are qualitatively similar. The results are presented for all content and for only long-tailed content (defined as objects with views $<$ 1,100). As Pull and TailGate rely on synthetic reads, we simulate multiple instances and report means, along with the standard deviations.

Performance of TailGate TailGate always outperforms Push in terms of the 95th percentile bandwidth across all PoPs in all scenarios. In the case of full information and perfect reads; TailGate reduces the 95th percentile bandwidth of Push upto 75% and upto 28% of Pull. For specifically long-tailed content, this number still grows: upto 88% of Push and 30% of Pull.

Effect of information quality There are two types of inaccuracies: one in read patterns when we go from per user reads to generic reads; the other is when we do not know the fraction of friends who will request content. When we look at the former, we note that though TailGate still performs better than Push (when we consider (Full Information,Imperfect Reads)), by upto 40%. However, TailGate has similar bandwidth costs as Pull (that has accurate information, hence is reported once). We then looked into the penalties. For Pull, the aggregated penalty is 126,846, while for TailGate under (Full Information,Imperfect Reads), it is 12,044. This happens at each considered location as shown in Fig. 6 which report the penalties paid by Pull and TailGate under (Full Information,Imperfect Reads). We remark that Push and TailGate under (Full Information, Perfect Reads) pay no penalty by design. In other words, though the effect of inaccurate reads is comparable bandwidth costs, the penalties are still 10 times lower than for Pull. Similar results for LT datasets (Pull: 95,087 vs TailGate: 9,162), as depicted in Fig. 6 specifically for each location. Taken together with the conservative nature of the evaluation, we believe TailGate is highly competitive to Pull, in terms

of bandwidth costs and latency.[7]

When we consider the inaccuracy in parameter p, we see that we need $p \geq 0.4$ to surpass Pull in terms of bandwidth costs. In other words, TailGate handles the inaccuracy in the click-rates much better than inaccuracy in read patterns. Moreover, TailGate still outperforms Pull in terms of penalties by almost 12 times even when the available read pattern is affected by an error of 2 hours (not shown in the paper).

Where do improvements come from? Under the considered scenario (4 PoPs), we find that knowledge of the social graph provides modest improvements – when we consider (Full Information,Imperfect Reads) and (No Information,Imperfect Reads), the only extra information used is the social network and we find a modest increase in the bandwidth costs as data is being uploaded to all PoPs instead of to only PoPs that have friends. However, this will change if we consider more PoPs. As for the accuracy of reads, it plays a stronger role in reducing costs.

In the supplemental material we present a study of the case where TailGate has *little* access to social information (NI, IR), but can help with QoE in the case of long-tailed YouTube videos [10], [21], [23]. Our experimental results show that TailGate increases the QoE by a factor 2, especially for long-tailed content.

8 Deployment Scenarios

OSN running our systems: An OSN provider such as Facebook can run TailGate. In this case, all the necessary information can be provided and as shown, TailGate provides the maximum benefit. The distributed architecture we consider throughout the paper is different from that employed currently by Facebook that operates three data centers (one being the master) and leases space at other centers [4]. However, we believe that large OSNs will eventually gravitate to the distributed architecture we described in Sec. 2.1, for reasons of performance and reliability mentioned in Sec. 1, as well as recent work that has shown that handling reads/writes out of one geographical site can be detrimental to performance for an OSN [29], pointing to an architecture that relies on distributed state. If the OSN provider leases bandwidth from external providers, our solutions decrease costs. If the provider owns the links, then they make optimal use of the link capacity, delaying equipment upgrades as links are provisioned for the peak.

CDNs with social information: Systems like CDNs are in general highly distributed (e.g., Akamai), but the architecture we used in this paper captures fundamental characteristics such as users being served out of the nearest PoP [8]. Existing CDN providers might not get access to social information, yet might be used by existing OSN providers to handle content. We have shown that even with limited access, the CDN provider can still optimize for bandwidth costs after making assumptions about access patterns.

CDNs without social information: Even without access to OSN information, a CDN can access publicly available information (e.g., Tweets) and use it to improve performance for its own customers.

7. Observe that the number of penalties paid by TailGate under (No Information, Imperfect Reads) scenario is the same as in (Full Information,Imperfect Reads).

9 Related Work

Distribution of long-tailed content has been addressed by several works, mostly in P2P networks [30], [31]. However, such swarm systems need extra resources (by way of replicates) and, as such, do not address transit bandwidth costs or latency constraints explicitly – requirements that TailGate addresses.

The popularity of OSNs has led to work that exploits social networking information to better inform system design. Solutions presented in this paper are, in part, motivated by findings presented by Wittie *et al.* [29] where the authors analyze the current Facebook architecture and uncover network performance problems the architecture introduces, including high bandwidth utilization and large delays. Distributing states to improve performance, based on geography or via clustering users on a social graph, has been explored in [32].

Recent work combines information from OSNs to improve CDN service [33], hence is similar in motivation to our system. The authors propose a similar mechanism to Buzztraq [34], wherein social cascades can be used to place content close to users. We also place content close to users, yet our focus is on *when* to distribute such content to minimize bandwidth costs (i.e., we consider a more realistic cost model). Our social-aware schedulers can be used along with the approach proposed in [33] that answers *where*. Similarly, in [35] the authors predict the popularity of videos and the origins of the views by analyzing the trends on micro-blogging sites, in order to optimize the video placement. Work by Laoutaris *et al.* [11] describes NetStitcher, a system that aims to do bulk transfers between data centers, by exploiting off-peak hours and storage in the network to send bulk data. NetSticher operates at the network layer, whereas we rely on information about access patterns at the application layer. Hence they are complementary.

In [36], the authors tackle a similar problem, keeping users' event feeds up to date through a combination of pushes from low-rate producers and pulls from high-rate producers. The main difference with our approach is that their aim is to minimize cumulative cost in terms of CPU or I/O and keeping latency at an acceptable level for users while we aim at reducing peak replication traffic (therefore traffic cost) and keeping latency at an acceptable level for users through the geo-diversified nature of our system.

10 Conclusion

Handling delivery of long-tailed content is a difficult task made harder with the wide proliferation of OSNs and geo-diversification of the underlying architecture.

We have proposed a social-aware scheduler, and its lightweight online adaptation to solve the above problem. They exploit information from social networks, e.g., the social graph and the regularity of activity patterns, to distribute long-tailed content while decreasing bandwidth costs.

With regard to the social-aware scheduler, our findings are: (1) the total WAN traffic is decreased by 55% by not sending updates to PoPs where they are not accessed; and (2) under the same monetary budget the social-aware scheduler improves information freshness, by several orders of magnitude (w.r.t. our penalty-based metric), over FIFO scheduling of updates. Such results encouraged us towards the development of an online solution named TailGate.

Using large traces gleaned from an OSN, we have shown that TailGate can reduce costs by as much as 80% over a naive FIFO based mechanism and as much as 30% over a pull-based approach that is employed by CDNs. Even in limited information scenarios, TailGate performs as well as Pull but reduces latency by 10X over Pull. In addition, we develop and deploy a simple prototype of TailGate on PlanetLab and show that it can help reduce end-user latency for long-tailed content. Taken together with bandwidth cost savings, and the fact that TailGate is lightweight, we envision it as complementary to existing CDN technologies.

11 Acknowledgements

Part of this work appeared in the proceedings of WWW 2012 [37]. This work has been partially supported by the FP7 project ENVISION of the EU.

References

[1] C. Anderson, *The Long Tail: Why the Future of Business is Selling Less of More*. Hyperion, 2006.
[2] B. Ager, F. Schneider, J. Kim, and A. Feldmann, "Revisiting Cacheability in Times of User Generated Content," in *GI*, 2010.
[3] "Twitter plans major data-center expansion," http://www.datacenterknowledge.com/archives/2013/05/10/twitter-plans-huge-data-center-expansion, visited Feb 2014.
[4] "Facebook Data Center FAQ," http://www.datacenterknowledge.com/the-facebook-data-center-faq, visited June 2013.
[5] "Facebook Hosts More Photos than Flickr and Photobucket," http://www.tothepc.com/archives/facebook-hosts-more-photos-than-flickr-photobucket/, visited June 2013.
[6] Facebook, "Facebook Ranked Second Largest Video Site," http://vator.tv/news/2010-09-30-facebook-ranked-second-largest-video-site, visited June 2013.
[7] R. I. M. Dunbar, "Neocortex Size as a Constraint on Group Size in Primates," *Journal of Human Evolution*, vol. 22, no. 6, pp. 469–493, 1992.
[8] C. Huang, A. Wang, J. Li, and K. W. Ross, "Measuring and Evaluating Large-Scale CDNs," in *IMC*, 2008.
[9] G. Linden, "Marissa Mayer at Web 2.0," http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html, visited June 2013.
[10] R. Torres, A. Finamore, J. R. Kim, M. Mellia, M. M. Munafo, and S. Rao, "Dissecting Video Server Selection Strategies in the YouTube CDN," in *ICDCS*, 2011.
[11] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez, "Inter-Datacenter Bulk Transfers with NetStitcher," in *SIGCOMM*, 2011.
[12] Forrester Consulting, "The Future of Data Center Wide Area Networking," http://infineta.broadchoice.com/orphan/forrester_report_download, visited June 2013.
[13] D. Beaver, S. Kumar, H. C. Li, J. Sobel, and P. Vajgel, "Finding a needle in Haystack: Facebook's photo storage," in *OSDI*, 2010.
[14] F. Benevenuto, T. Rodrigues, M. Cha, and V. Almeida, "Characterizing user behavior in online social networks," in *IMC*, 2009.
[15] F. Schneider, A. Feldmann, B. Krishnamurthy, and W. Willinger, "Understanding online social network usage from a network perspective," in *IMC*, 2009.
[16] "Burstable Billing," http://en.wikipedia.org/wiki/Burstable_billing, visited June 2013.
[17] X. Dimitropoulos, P. Hurley, A. Kind, and M. P. Stoecklin, "On the 95-percentile billing method," in *PAM*, 2009.
[18] F. Schneider, A. Feldmann, B. Krishnamurthy, and W. Willinger, "Understanding Online Social Network Usage from a Network Perspective," in *IMC*, 2009.
[19] J. Hamilton, "The cost of latency," http://perspectives.mvdirona.com/2009/10/31/TheCostOfLatency.aspx, visited October 2013.
[20] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, and A. Wolman, "Volley: Automated Data Placement for Geo-Distributed Cloud Services," in *NSDI*, 2010.
[21] "YouTube CDN Architecture," Private Communication, Content Delivery Platform, Google.
[22] J. Hamilton, "Inter-datacenter replication and geo-redundancy," http://perspectives.mvdirona.com/2010/05/10/InterDatacenterReplicationGeoRedundancy.aspx, visited June 2013.
[23] highscalability, "YouTube Architecture," http://highscalability.com/youtube-architecture, visited June 2013.
[24] J. M. Pujol, V. Erramilli, G. Siganos, X. Yang, N. Laoutaris, P. Chhabra, and P. Rodriguez, "The Little Engines that Could: Scaling Online Social Networks," in *SIGCOMM*, 2010.
[25] H. Kwak, C. Lee, H. Park, and S. Moon, "What is Twitter, a Social Network or a News Media?" in *WWW*, 2010.
[26] R. W. Sinnott, "Virtues of the Haversine," *Sky and Telescope*, vol. 68, p. 159, 1984.
[27] V. Erramilli, X. Yang, and P. Rodriguez, "Explore what-if scenarios with SONG: Social Network Write Generator," http://arxiv.org/abs/1102.0699, 2011.
[28] R. Crane and D. Sornette, "Robust dynamic classes revealed by measuring the response function of a social system," *PNAS*, vol. 105, no. 41, pp. 15 649–15 653, 2008.
[29] M. P. Wittie, V. Pejovic, L. Deek, K. C. Almeroth, and B. Y. Zhao, "Exploiting Locality of Interest in Online Social Networks," in *CoNEXT*, 2010.
[30] R. S. Peterson and E. G. Sirer, "AntFarm: Efficient Content Distribution with Managed Swarms," in *NSDI*, 2009.
[31] D. S. Menasche, A. A. Rocha, B. Li, D. Towsley, and A. Venkataramani, "Content Availability and Bundling in Swarming Systems," in *CoNEXT*, 2009.
[32] T. Karagiannis, C. Gkantsidis, D. Narayanan, and A. Rowstron, "Hermes: Clustering Users in Large-Scale E-mail services," in *SoCC*, 2010.
[33] S. Scellato, C. Mascolo, M. Musolesi, and J. Crowcroft, "Track globally, deliver locally: Improving content delivery networks by tracking geographic social cascades," in *WWW*, 2012.
[34] N. Sastry, E. Yoneki, and J. Crowcroft, "Buzztraq: Predicting Geographical Access Patterns of Social Cascades Using Social Networks," in *SNS*, 2009.
[35] Z. Wang, L. Sun, C. Wu, and S. Yang, "Guiding Internet-Scale Video Service Deployment Using Microblog-based Prediction," in *INFOCOM*, 2012.
[36] A. Silberstein, J. Terrace, B. F. Cooper, and R. Ramakrishnan, "Feeding frenzy: Selectively materializing users' event feeds," in *SIGMOD*, 2010.
[37] S. Traverso, K. Huguenin, I. Trestian, V. Erramilli, N. Laoutaris, and K. Papagiannaki, "TailGate: Handling Long-Tail Content with a Little Help from Friends," in *WWW*, 2012.

Stefano Traverso earned his Ph.D. degree in Networking Engineering at Politecnico di Torino, Italy, in 2012. His research interests include P2P networks, overlay networks, network measurement, and content delivery networks. During his Ph.D. and Post-doc he has been visiting Telefonica I+D research center in Barcelona, Spain, and NEC Laboratories in Heidelberg, Germany. He is currently a Post-doc Fellow of TNG group of Politecnico di Torino.

Kévin Huguenin earned his M.Sc. degree from Ecole Normale Supérieure de Cachan and the Université de Nice – Sophia Antipolis, France, in 2007 and his Ph.D. in computer science from the Université of Rennes, France, in 2010. His research interests include performance, security and privacy in networks and distributed systems. He is currently a Post-Doctoral Researcher at EPFL, Switzerland, in the LCA group.

Ionut Trestian earned his B.Sc. degree in Computer Science from the Technical University of Cluj-Napoca, Romania, in 2007. He recently earned his Ph.D degree at Northwestern University. His research interests include network measurement, network security, overlay networks and social networks. He currently works as a Software Engineer at Amazon.

Vijay Erramilli earned the Ph.D. degree in Computer Science from Boston University, Boston, MA, in 2008. He then joined Telefonica Research, Barcelona, Spain, in January 2009, where he is currently an Associate Researcher.

Nikolaos Laoutaris earned the Ph.D. degree in Computer Science from the University of Athens, Athens, Greece, in 2004. He then went to Boston University, Boston, MA, for a post-doctoral position, and later Harvard University, Cambridge, MA. He joined Telefonica Research, Barcelona, Spain, in 2007 and is currently a Senior Researcher.

Konstantina Papagiannaki earned the first degree in electrical and computer engineering from the National Technical University of Athens (NTUA), Athens, Greece, in 1998, and the Ph.D. degree from the Computer Science Department of University College London (UCL), London, U.K., in 2003. She is the scientific director of the Internet Systems and Networking scientific group at Telefonica I+D in Barcelona.

Supplemental File: Social-Aware Replication in Geo-Diverse Online Systems

Stefano Traverso, Kévin Huguenin, *Member, IEEE,* Ionut Trestian, *Student Member, IEEE,* Vijay Erramilli, Nikolaos Laoutaris, and Konstantina Papagiannaki

1 Budget Allocation

Algorithm 1 Budget allocation across K PoPs

Input: B ▷ Total budget
Output: $\mathbf{b}$ ▷ Budget allocation

$b \leftarrow 0$ ▷ Total budget allocated
$\mathbf{b} \leftarrow (0, \ldots, 0)$ ▷ Budget allocation
while $b < B$ **do**
  $p_{\min} \leftarrow \infty$ ▷ Minimum penalty
  $k_{\min} \leftarrow \perp$
  for $k \in \{1, \ldots, K\}$ **do**
    $\mathbf{b}_k \leftarrow \mathbf{b}_k + \delta_b$ ▷ Allocates a budget increment to the k-th PoP
    $p \leftarrow \text{penalty}(\mathbf{b})$ ▷ Computes the penalty for a given budget allocation
    $\mathbf{b}_k \leftarrow \mathbf{b}_k - \delta_b$
    if $p < p_{\min}$ **then**
      $p_{\min} \leftarrow p$ ▷ Update minimum penalty
      $k_{\min} \leftarrow k$
    end if
  end for
  $\mathbf{b}_{k_{\min}} \leftarrow \mathbf{b}_{k_{\min}} + \delta_b$ ▷ Allocate the budget to the PoP s.t. the penalty is minimized
  $b \leftarrow b + \delta_b$
end while

Where B is the total budget to be allocated to the PoPs, $\mathbf{b}$ is the budget allocation (i.e., a vector of real values where the k-th element of $\mathbf{b}$ is the budget allocated to the k-th PoP), and δ_b is the budget increment for the iterative algorithm (i.e., the budget increment to be allocated at each iteration of the allocation algorithm). The variable b is the total budget allocated so far, and $p_{\min}$ denotes the minimum penalty obtained, at a given iteration, for all the possible allocations of the budget increment δ_b (and $k_{\min}$ denotes the PoP to which the budget increment is assigned in the allocation that achieves the minimum penalty).

2 Data details

We rely on a large dataset of 41.7M users with 1.47B edges obtained through a massive crawl of Twitter between June – Sept. 2009 [1]. We then collected the users' location information by conducting our own crawl and translated everything to latitude/longitude using Google Maps API. In the end, we extracted locations for 8,092,624 users from about 11M users that had entered location information. We use this social graph only between these nodes for our analysis in this paper. With regards to the user's locations in the dataset, we find that US has the most users (55.7%), followed by UK (7.02%) and Canada (3.9%).

2.1 Upload Activity

For the users who had entered their locations, we collected their tweets. We found that the mean number of tweets was 42 per user. Not all users had tweet activity; the number of active users (who tweeted at least once) was 6.3M users. For these 6.3M users, we ended up collecting approximately 499M tweets, until Nov. 2010.

This dataset is valuable for characterizing activity patterns of users. From these tweets, we extracted those that contain hyperlinks pertaining to pictures (plixi, Twitpic, etc.) and videos (Youtube, Dailymotion, etc.) that we consider as UGC[1], which gave us 101,079,568 links. Here, we consider that posting a URL pointing to a video is equivalent to uploading the same video as in a social network that offers the option to host multimedia content (e.g., Facebook for photos), a user would have uploaded her generated content directly instead of uploading it to a third party service and posting a URL pointing to it.

We recorded the size of each piece of content that is shared, resolving URL shorteners when needed. The largest file happened to be of a cricket match on YouTube, with a size 1.3G on 480p (medium quality). We collected the number of views for each link, wherever available, and the closest (Kullback-Leibler (KL) distance) fit was the log-normal distribution (parameters: (10.29,3.50)); we found around 30% of the content to be viewed less than 500 times. The most popular was a music video by Lady Gaga on YouTube, viewed more than 300M times.

2.2 Geo-distributed PoPs

To study the effects of geo-diversity on bandwidth costs, we use location data and assign users to PoPs

1. Note that the popularity distribution of such content exhibit a shorter tail [2], [3] than the content uploaded on highly-social platforms on which users upload content for a more limited audience, e.g., pictures of events with friends or family members.

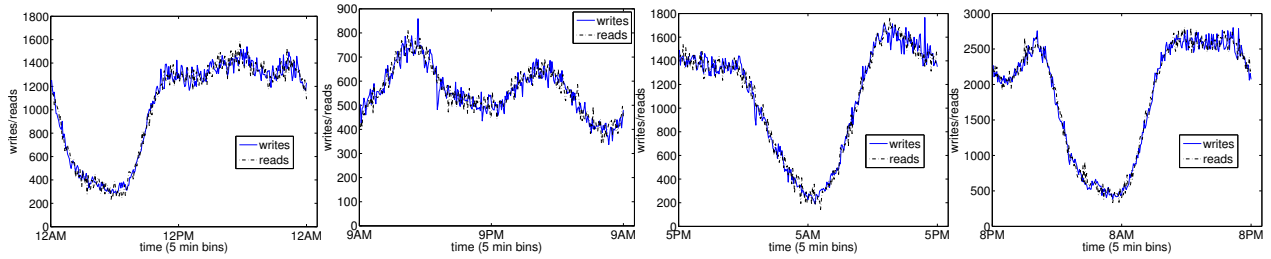

Fig. 1. Writes along with synthetic reads (a) London (b) Tokyo (c) LA (d) Boston.

distributed around the world. We assume the distributed architecture as described in the preliminaries. Because we simulate the actual effects of different PoP locations, for the social-aware scheduler we consider multiple placements. The number of PoPs is a free variable but the locations we choose are fixed as we do not address the PoP placement problem here. The results are shown in Table 1. Because of the high number of users that are located in the US, several of the PoPs will cover this region and this also explains the imbalance observed in the number of users for the 2 PoPs case. Note that for the above 3 PoPs, the number of users is relatively balanced across them.

City	2 PoPs	3 PoPs	4 PoPs	5 PoPs	6 PoPs	7 PoPs
Boston	5,608,894	3,476,676	2,187,867	1,318,388	1,318,388	1,318,388
London	2,483,730	2,274,409	2,274,406	2,274,374	1,684,098	1,492,839
LA		2,341,539	1,569,944	1,563,705	1,267,445	1,267,445
Houston			2,060,407	1,454,755	1,454,755	1,454,455
Chicago				1,481,402	1,481,366	1,481,366
Tokyo					886,572	533,166
Delhi						544,965

TABLE 1
Users per given PoP for different layouts.

For TailGate, we assume there exist data centers in these four locations: Boston, London, LA and Tokyo. We assign users to locations by using a simple method: compute the distance of a user to a location, and assign the user to the nearest location (w.r.t. the Haversine distance). For the four locations, we observe the following distribution of users: (Boston: 3,476,676, London: 1,684,101, LA: 2,045,274, Tokyo: 886,573). The East Coast of the US dominates in our datasets. The relatively low number of users in Asia is due to the fact that most users in Asia prefer local versions of OSNs. However, we choose Tokyo precisely for this point – users in Asia comprise social contacts of users from around the world, sharing and requesting content, and adding to bandwidth costs. We find that, on average, a user has 19.72 followers in her own PoP and 8.91 followers in each of the other PoPs. It is well known that friends in social networks are located close together with respect to geographical distance [4].

2.3 Read Activity

Our systems rely on information about accesses, i.e., reads. The ideal information will be about *who* requests the content and *when*. We could not obtain direct read patterns from Twitter/FB, so we proceeded as follows.

To get an idea on *who* requests, we collected packet traces via TCPDump from an outgoing link that connected a university in northern Italy (9th Mar, 2011) to the rest of the Internet. The collection was carried out for two distinct periods: from 11AM to 4PM and from 10AM to 6PM. We extracted `HTTP-POST`, `HTTP-GET` requests that correspond to the Facebook domain. We further collected all links corresponding to pictures and videos that show up on users' wall feeds. The clicks on these icons lead to new `HTTP-GET` requests, that are counted as requests for the content. We found that, on average, 9.8% of the posted links were clicked on. It was hard for us to obtain a per-user metric, as the users were behind a NAT device. We looked at smaller set of 200 users that were not behind a NAT device, and observed a similar 10% click rate. Note that in some social networks, users might exhibit much more selective behaviors, varying from one relationship to another (as shown in [5], [6]) while in some cases (e.g., when using a thick client) the content might be downloaded automatically. Therefore, the click-through rate might be very different from one application context to another. We use the observed rate as the probability of a friend requesting content (ρ) for our evaluation, when needed.

Note that future reads are impossible to know, but due to strong regularity in user behavior [7], an OSN such as Facebook or Twitter can predict future accesses with high accuracy. For the purposes of this work – we make the assumption that read patterns follow a diurnal trend similar to write/upload patterns, as observed by other authors [8], [9]. In order to generate read patterns at a fine time scale (seconds), we use SONG [10], which we also describe in the Appendix.

For TailGate, we are interested in evaluating different scenarios, as well as studying the affect of quality of information on performance gains from TailGate. In order to do this, we generate *two* different types of

read patterns: the first type is when social information is available at a fine scale, for instance when Facebook is operating TailGate; and the second is when little or no social information is available, for instance when a CDN is operating TailGate.

Reads with fine grained information: Each user is assigned reads. For this, we make the assumption that the distribution of the number of reads per user follows the same distribution as the number of uploads per user; a log-normal distribution.

Reads with no social information: We assume we do not have enough information to predict reads accurately, but rather general trends, such as diurnal trends, are known. For this, once we generate reads over time bins, we normalize all bins with the total number of reads found across all bins. In other words, we create a diurnal trend for reads. In Fig. 1, we plot the update patterns given by the data and synthetic reads generated for the **day** dataset for all the four centers we consider. Note that the updates follow a diurnal trend and the synthetic reads follow a similar pattern.

Finally, we distinguish long-tailed contents as those which were requested less than 1,100 times (roughly corresponding to 70% of the contents available in our trace). This choice was driven by a modal shift we observed around this value in our dataset. We however evaluated the sensitivity of our results with respect to this choice, varying the threshold for long-tailed content from 500 to 4,000, but we could not appreciate any significant variation in TailGate performance.

2.4 Limitations and Assumptions

We note here that Twitter now has around ∼200M users[2] and our dataset is a relatively small sample. Hence all numbers presented in this paper should be interpreted in this context. An ideal dataset for evaluation would be the UGC that is uploaded and shared on an OSN, but such data is not available to us. Instead, we use the links collected above as proxy for the media, which can be shared over OSNs, and we recognize that activity patterns with regards to posting and sharing media content should follow similar diurnal patterns [9]. We assume read patterns follow a diurnal trend similar to write patterns [8], [9]. Note that we are more interested in time-of-day effects; more sophisticated read patterns where we consider content interest, quality of content, etc. can be used, but we do not have sufficient information to calibrate these effects. We also assume a simple homogeneous click through model in which all users click on the links shared by their friends with a fixed probability, constant across users. In addition, we assume that all the contents uploaded by the users are unique, which is not the case when users can re-share contents shared by their friends (e.g., on Facebook).

2. http://en.wikipedia.org/wiki/Twitter

	ALL		POP		LT	
	d_time1	d_time2	d_time1	d_time2	d_time1	d_time2
Boston	4.11 (6.63)	4.81 (8.51)	3.69 (6.61)	4.39 (7.67)	4.73 (6.67)	4.68 (8.35)
LA	2.94 (5.53)	2.70 (5.71)	1.59 (2.55)	1.72 (3.33)	4.70 (7.46)	3.49 (6.50)
Lon	5.15 (5.06)	4.56 (5.11)	4.57 (3.53)	4.06 (3.37)	6.43 (6.72)	4.88 (5.92)
Tokyo	6.18 (6.76)	5.09 (5.64)	5.36 (6.23)	5.10 (5.42)	6.88 (6.67)	4.65 (4.27)

TABLE 2
Average download times (in seconds) with standard deviations for buffering stages.

3 Case Study: Long-Tailed Videos on YouTube

In this section, we study the case where TailGate has *little* access to social information (NI, IR), but can help with QoE in the case of long-tailed YouTube videos [11], [12], [13]. The entity controlling TailGate (e.g., CDN) can rely on publicly available information (e.g., Tweets), as we do here, and use TailGate to request or "pull" content to intelligently prefetch content before the real requests for the content, thereby decreasing latency for their customers. Towards this end, we develop a simple prototype of TailGate based on the design, and we deploy it on 4 PlanetLab nodes at the same locations: Boston, London, LA and Tokyo.

We proceed as follows: we rely on our dataset, where we assign the four sets of users to different "PoPs" as given by Planetlab nodes. We extract the set of links that correspond to YouTube videos from our dataset, along with the times they were posted. Note that the information related to the properties of YouTube videos accessed from Twitter is public: anyone can collect this information. We then provide this set of writes as input to TailGate, assuming no social information and that the expected reads in various locations follow a diurnal pattern. We get a schedule as output from TailGate, which effectively schedules transfers between the four locations. We take this schedule and instead directly *request* the YouTube videos from various sites, at a time given by TailGate, thus emulating transfers.

After this, we request the videos at the time of the "read", that is, we "emulate" users from each location issuing read requests for each video by sampling from the diurnal trend. Therefore each video is requested twice: the first time for emulating the transfer using a schedule given by TailGate, and the second time, for emulating a legitimate request by a user to quantify the benefit. Note that the first request would also be emulating a PULL, as we emulate a cold-miss. Hence any improvements we notice would be an improvement over PULL. We use `wget` with the `-no-cache` option for all our operations, to limit caching effects.

We focus on the Quality of Experience (QoE) for the end-user. In order to measure this, we first look at the proportion of a file that is downloaded during the initial buffering stage, after which the playout of the video is smooth. We say the playout is smooth

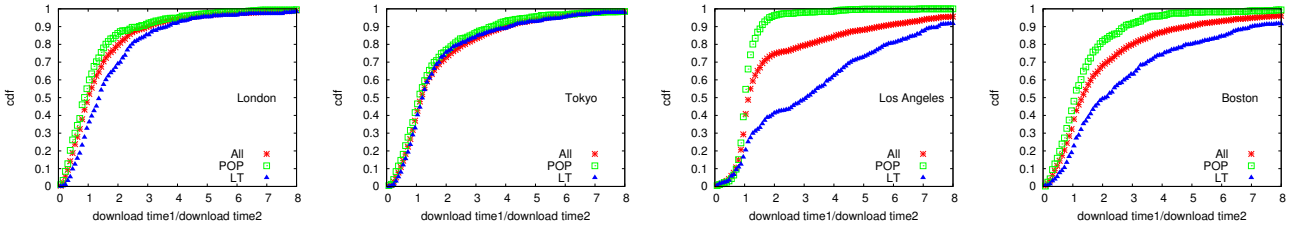

Fig. 2. Performance figures for YouTube videos, improvements for download times for buffering stage.

if the download rate[3] for a file drops at 70% of the original rate. We tested other values and had similar results. We found that, on average, the playback is smooth after 15% of a file is downloaded. Therefore we noted the delay in terms of the time it takes for the first 15% of a file to be downloaded. As we download each video twice, once at a time given by TailGate and the second as representing the actual read request, we measure both and plot the cdfs of ratios (dload_time1/dload_time2) in Fig. 2. We plot for three different cases: "all" is the entire dataset; "pop" stands for only those videos that are popular ($\geq$ 500K views); and "LT" stands for long-tailed videos ($\leq$ 1,100 views– we observed a mod shift around this value in our dataset). First, we note that there is an improvement of a factor of 2 and it is higher for at least 30% of the videos for all locations. Second, this improvement is even more pronounced for "LT" videos, highlighting that TailGate aids long-tailed content. Indeed, we remark that popular content has in general a larger probability of being present at the PoP/cache at the moment of the first request, thus lowering de facto the benefits of adopting TailGate. This does not hold for long-tailed videos that show a lower probability of being replicated at many PoPs, thus increasing the potential gain of TailGate. Observe that given the bulky nature of video content, downloading a small fraction of a video may take some tens of seconds, so that the gain in time achieved by TailGate is significant for the QoE perceived by users. In Table 2, we report the mean and standard deviations of download times for all considered scenarios. For some videos, we see a decrease in performance (dload_time1/dload_time2 < 1). This could be due to load-balancing. In fact for Tokyo, we found that the closest PoP for YouTube seems to be relatively far (Korea) in the first place.

APPENDIX

We briefly review the model we use for generating synthetic reads. Let $X_i(t)$ denote the number of reads produced by user i, $1 \leq i \leq N$ with N the total number of users at a time instant t, where $X(t) = \sum_{\forall i} X_i(t)$. The time can vary from seconds to weeks. The description $X(t), \forall t$ gives the time series aggregated over all users. We need to account for two different time-scales - the first time scale spans multiple hours or days and we note the presence of diurnal trends. The second time scale spans seconds to a couple of hours where the mean and the variance are fairly stable. For the first time scale, we can have a model for the mean m_t of the time series that varies with time in a predictable way. For the second time scale, we can have a stochastic component. The model then is

$$X(t) = m_t + \sqrt{am_t}W_t \quad (1)$$

where m_t is function of time and W_t is a stochastic component which can be a zero-mean, finite variance process and a is a parameter called 'burstiness' (with the units:reads-secs) that accounts for magnitude of fluctuations.

The main method for generating synthetic reads is as follows: First we generate m_t using a Fourier series by first extracting the largest Fourier coefficients of the write time-series, then we add appropriately scaled noise, by estimating a, to the diurnal trend at each time bin (in our case, seconds). For our **day** and **week** datasets, we use the top 5 Fourier coefficients to generate the diurnal pattern and used WGN with an appropriate scale parameter to generate the reads. The generated time series contains the number of reads in a given time bin.

3. To measure the download throughput, we employed the download rate estimator embedded in `wget`.

REFERENCES

[1] H. Kwak, C. Lee, H. Park, and S. Moon, "What is Twitter, a Social Network or a News Media?" in *WWW*, 2010.
[2] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon, "I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system," in *IMC*, 2007.
[3] N. R. Sastry, "How to tell head from tail in user-generated content corpora," in *ICWSM*, 2012.
[4] D. Liben-Nowell, J. Novak, R. Kumar, P. Raghavan, and A. Tomkins, "Geographic Routing in Social Networks," *PNAS*, vol. 102, pp. 11 623–11 628, 2005.
[5] C. Wilson, B. Boe, A. Sala, K. P. Puttaswamy, and B. Y. Zhao, "User interactions in social networks and their implications," in *Eurosys*, 2009.
[6] H. Chun, H. Kwak, Y.-H. Eom, Y.-Y. Ahn, S. Moon, and H. Jeong, "Comparison of online social relations in volume vs interaction: a case study of cyworld," in *IMC*, 2008.
[7] S. A. Golder, D. M. Wilkinson, and B. A. Huberman, "Rhythms of Social Interaction: Messaging Within a Massive Online Network," in *C&T*, 2007.
[8] F. Benevenuto, T. Rodrigues, M. Cha, and V. Almeida, "Characterizing User Behavior in Online Social Networks," in *IMC*, 2009.

[9] F. Schneider, A. Feldmann, B. Krishnamurthy, and W. Willinger, "Understanding Online Social Network Usage from a Network Perspective," in *IMC*, 2009.
[10] V. Erramilli, X. Yang, and P. Rodriguez, "Explore what-if scenarios with SONG: Social Network Write Generator," http://arxiv.org/abs/1102.0699, 2011.
[11] highscalability, "YouTube Architecture," http://highscalability.com/youtube-architecture, visited June 2013.
[12] R. Torres, A. Finamore, J. R. Kim, M. Mellia, M. M. Munafo, and S. Rao, "Dissecting Video Server Selection Strategies in the YouTube CDN," in *ICDCS*, 2011.
[13] "YouTube CDN Architecture," Private Communication, Content Delivery Platform, Google.