TailGate: Handling Long-Tail Content with a Little Help from Friends

Stefano Traverso
Politecnico di Torino
stefano.traverso@polito.it

Kévin Huguenin
EPFL
kevin.huguenin@epfl.ch

Ionut Trestian
Northwestern University
ionut@northwestern.edu

Vijay Erramilli
Telefonica Research
vijay@tid.es

Nikolaos Laoutaris
Telefonica Research
nikos@tid.es

Konstantina Papagiannaki
Telefonica Research
dina@tid.es

ABSTRACT

Distributing long-tail content is an inherently 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 (UGC) and online social networks (OSNs) create and reinforce such popularity distributions. Second, the recent trend of geo-replicating content across multiple PoPs spread around the world, done for improving quality of experience (QoE) for users and for redundancy reasons, can lead to unnecessary bandwidth costs.

We build TailGate, a system that exploits social relationships, regularities in read access patterns, and time-zone differences to efficiently and selectively distribute long-tail content across PoPs. We evaluate TailGate using large traces from an OSN and show that it can decrease WAN bandwidth costs by as much as 80% as well as reduce latency, improving QoE. We deploy TailGate on PlanetLab and show that even in the case when imprecise social information is available, TailGate can still decrease the latency for accessing long-tail YouTube videos by a factor of 2.

Categories and Subject Descriptors

H.3.4 [**Information Systems**]: Systems and SoftwarePerformance evaluation (efficiency and effectiveness)

Keywords

Social networks, Content Distribution, Long-Tail, Geo-Replication

1. INTRODUCTION

Online content distribution technologies have witnessed many advancements over the last decade, from large CDNs to P2P technologies, but most of these technologies are inadequate while handling unpopular or long-tailed content. By long-tailed we refer to the popularity of the content, in terms of accesses. CDNs find it economically infeasible to deal with such content – the distribution costs for content that will be consumed by very few people globally is higher than the utility derived from delivering such content [2]. Unmanaged P2P systems suffer from peer/seeder shortage and meeting bandwidth and/or 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) 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 such as Flickr [35], and they now host and serve a large proportion of videos as well [9]. Content created and shared on social networks is predominantly long-tailed with a limited interest group, specially if one considers notions like Dunbar's number [7]. The increasing adoption of smartphones, with advanced capabilities, will further drive this trend.

In order to deliver content and handle a diverse user-base [16, 23], most large distributed systems are relying on geo-diversification, with storage in the network [32, 10, 33]. One can push or prestage content to geo-diversified PoPs closest to the user, hence limiting the parts of the network affected by a request and improving QoE for the user in terms of reduced latency. However, it has been shown that transferring content between such PoPs can be expensive due to WAN bandwidth costs [21, 11]. For long-tailed content, the problem is more acute – one can *push* content to PoPs, only to have it not consumed, wasting bandwidth. Inversely one can resort to *pull*, and transfer content only upon request, but leading to increased latencies and potentially contributing to the peak load. Given the factors above, along with the inability of current technologies to distribute long-tail content [2, 18] while keeping bandwidth costs low, it would appear that the problem of distributing long-tailed content is and will be a difficult endeavor.

Our Contribution: In this paper, we present a system called TailGate that can distribute long-tailed content while lowering bandwidth costs and improving QoE. The key to distribution is to know (i) *where* the content will likely be consumed, and (ii) *when*. If we know the answers, we can *push* content where-ever it is needed, at a time before it is needed, and such that bandwidth costs are minimized under peak based pricing schemes like 95th percentile pricing [4]. 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 since network dimensioning in both cases depends on the peak bandwidth consumption. Recent proposals like NetSticher [21] have proposed systems to distribute content be-

WWW 2012, April 16–20, 2012, Lyon, France.
ACM 978-1-4503-1229-5/12/04.

tween geo-diversified centers, while minimizing bandwidth costs. TailGate augments such solutions by relying on a hitherto untapped resource – information readily available from OSNs.

More specifically, TailGate relies on the 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 consumption patterns of users. First, users follow strong diurnal trends while accessing data [30]. 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, and trying to spread and *flatten* out the traffic caused by moving content. The scheduling scheme enforces an informed *push* scheme (described in Sec 3), reduces peaks and hence the costs. In addition, the content is pushed to the relevant sites before it is likely accessed – reducing the latency for the end-users. We designed TailGate to be be simple and adaptable to different deployment scenarios.

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. Tailgate takes advantage of this information and we compare Tailgate's performance in terms of reduction in bandwidth costs (as given by a reduction in the 95th percentile) and improving QoE for the user under different scenarios and using real data. When compared against a naive *push*, we see a reduction 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 content, the improvement is even more. We vary the quality of information available to Tailgate and find even when TailGate has less precise information, TailGate still performs better than push and is similar to pull in terms of bandwidth costs, while lowering latency (improving QoE) for upto 10 times of the requests over pull. We show that even in an extreme setting where TailGate has lower access to information in a live setting, TailGate can reduce latency for the end-user to access long-tailed content by a factor of 2.

2. PRELIMINARIES

For the sake of exposition, we describe a generic distributed architecture that will provide the template for the design and analysis of TailGate. In Sec. 6, we show how this architecture can be used for different scenarios – OSN providers and CDNs. After describing the architecture, we provide a simple motivating example. At the end of the section we list the requirements that a system like TailGate needs to fulfill.

2.1 Architecture

We consider an online service having 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 are connected to each other by links. These links can be owned by the entity owning the PoPs (for instance, online service providers like Google, Facebook or a Telco-operated CDN) or the bandwidth on these links can be leased from network providers.

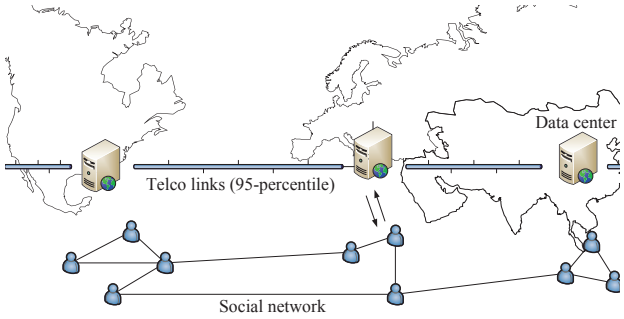

Figure 1: Generic distributed architecture: Servers or PoPs geo-distributed, handling content for geographically close users. Content created is first uploaded to geographically closest PoP, then this content is distributed to other PoPs

Users are assigned 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 like Volley [1]. 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 defacto mechanism (also known as a cold-miss) used by most CDNs [37]. We use this 'serve-if-available' else '*pull*-when-not available' mechanism for this paper as the baseline and we shall 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 is TailGate necessary: Toy Example

Consider user Bob living in Boston and assigned to the Boston PoP in Fig. 1. 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 US, Europe and Asia. These geographically distributed set of friends are assigned to the nearest PoP respectively. Bob logs in to the application at 6PM local time (peak time) and uploads a family video shot in HD that he wants to share. 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 may not be consumed at all thus having contributed to the 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. How-

ever, user activity follow 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. 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 – a loss for all.

TailGate's Approach: Instead of pushing content as soon as Bob uploads, 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, and 7AM is still earlier that 12PM when Alice is likely to log in. Therefore Alice can access Bob's content quickly, hence experience relatively high QoE. The provider has transferred the content during off-peak hours, decreasing costs – a win-win scenario for all. TailGate is built upon this intuition where such time differences between content being *uploaded* and content being *accessed* is exploited. In a geo-diverse system, such time differences exist anyway. In order to exploit these time differences, TailGate needs information about the social graph (Alice is a friend of Bob), where these contacts reside (Alice lives in London) and the likely access patterns of Alice (she will likely access it at 12PM).

2.3 System Requirements

TailGate needs to address and balance the following requirements:

Reduce bandwidth costs: Despite the dropping price of leased WAN bandwidth and networking equipment, the growth rate of UGC combined with the incorporation of media rich long tail content (e.g. images and HD videos) makes WAN traffic costs a big concern [14, 11]. For instance, the traffic volume produced by photos on Facebook can be in thousands of GB from just one region, e.g. NYC [36].

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

Online and reactive: The scale of UGC systems [15] 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 TailGate does not do

TailGate optimizes for bandwidth costs and does not consider storage constraints. It would be interesting to consider storage as well but we believe the relatively lower costs of storage puts the emphasis on reducing bandwidth costs [37]. TailGate does not deal with dynamic content like profile information etc. in OSNs as other systems do [27]. TailGate deals with large static UGC that is long-tailed and hence not amenable to existing distribution solutions.

3. TAILGATE

In this section, we first formulate the central problem that TailGate deals with – scheduling content updates to different PoPs in order to minimize bandwidth costs and latency, which we capture by way of a metric called "penalty". We then describe the algorithm TailGate uses that satisfies the requirements mentioned in Sec 2.3, along with existing solutions.

3.1 Formulation

Let $\{u_1, \ldots, u_N\}$ be the set of users and $\{S_1, \ldots, S_K\}$ the set of PoP sites, distributed at different locations around the world. As already discussed in Sec. 2.1, we assume each user is assigned to a site and the user's content is uploaded to this site. The friends of this user can be assigned to this site or to the other sites, depending on their location. The social network is captured by the set $F(u_n)$, consisting of social contacts of user u_n. We denote by $S(u_n)$ the master(closest) site of user u_n, $u_n \rightarrow S_k$ the fact that the user u_n needs to send data to site S_k.

We model the evolution of the system at discrete time bins $t \in [0, T]$ and we assume that bins are short enough – typically less than a minute – so that a user performs at most one read and at most one update during a time bin. Updates and reads performed by user u_n during time bin t are denoted respectively by $w_n^{[t]}$ and $r_n^{[t]}$ (binary matrices). We denote by $\mathbf{w}_n^{[t]}$ the actual size of the updates sent by user u_n during time bin t. We assume that upon a read operation a user can access the content posted by the user's friends. Table 1 summarizes the terminology used in our formulation.

The decision variable of the optimization problem latency vs. bandwidth costs is the update schedule between sites, denoted by $t_{n,k}^{[t]}$; the number of updates of user u_n sent to site S_k during time bin t. We denote by $\mathbf{t}_{n,k}^{[t]}$ the size of the updates of user u_n sent to site S_k during time bin t. Transmission of updates of a *given* user to a given S_k follows a FIFO policy to ensure temporal consistency. Hence the missing updates are always the most recent ones.

Optimization metric: Bandwidth costs The incoming and outgoing traffic volumes of each site S_k depends on the upload strategy and updates:

$$v_k^{\downarrow[t]} = \sum_{k' \neq k} \left(\sum_{u_n \rightarrow S_{k'}} \mathbf{t}_{n,k}^{[t]} \right) \quad (1)$$

$$v_k^{\uparrow[t]} = \sum_{k' \neq k} \left(\sum_{u_n \rightarrow S_k} \mathbf{t}_{n,k'}^{[t]} \right) \quad (2)$$

In general, a peak-based pricing scheme is used as a cost function ($p_k(\cdot)$). The most common is the 95th percentile ($q(\cdot)$) of the traffic volume (typically a stepwise concave function). Therefore the bandwidth costs incurred at site S_k is $c_k = p_k(\max(q(v_k^{\downarrow}), q(v_k^{\uparrow})))$ and the total bandwidth cost is the sum of all the c_k.

Constraint: Latency via Penalty metric In order to capture the notion of latency, which is closely related to a "cold-miss" at a site (as discussed in Sec. 2.1), for a user u_n at site S_k is captured by the number $d_{n,k'}^{[t]}$; updates of u_n that are missing at site S_k:

$$d_{n,k}^{[t]} = \begin{cases} \sum_{t'=0}^{t'=t} w_n^{[t]} - t_{n,k}^{[t]} & \text{if } S(u_n) \neq S_k \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

which is representative of the number of times the content has to be fetched from the server where it is originally hosted, increasing latency. To evaluate the perceived latency, we define a penalty system: every time a user requests one of her friends' updates and it is not available, the total penalty is incremented by the number above. The total penalty P is:

$$P(T) = \sum_{u_n \in \mathcal{U}} \left(\sum_{u_m \in F(u_n)} d^{[t]}_{n,S(u_m)} \cdot r^{[t]}_m \right) \quad (4)$$

Handling real-world constraints: In the online case, reads are not known in advance and the algorithm must therefore be oblivious of the input read matrix $r^{[t]}_n$. Moreover, an OSN might be hesitant to release individual read patterns and the social graph to the entity (CDN) operating TailGate. For these reasons, we replace the reads in the above problem with estimated reads – a generic diurnal pattern. Assuming that each user of the social network has a regular read activity, captured by a probability $\rho^{[t]}_n$ of user u_n performing a read operation during time bin t, then the probability of an update posted at time t by user u_n to be read at time t' on server S_k: $\rho^{[t]}_{n,k} = 1 - \prod_{u_m \to F_n \cap S_k} (1 - \rho^{[t]}_m)$. We can therefore derive an expression of the expected read, which is the deadline $\delta^{[t]}_{n,k}$ (*i.e.*, first read) on server S_k of an update posted by u_n at time t and this replaces $r^{[t]}_n$:

$$\delta^{[t]}_{n,k} = \sum_{t'=t}^{+\infty} \left(t' \cdot \rho^{[t']}_{n,k} \cdot \prod_{t''=t}^{t'-1} (1 - \rho^{[t']}_{n,k}) \right) \quad . \quad (5)$$

T	Number of time intervals in a charging period, $t \in T$.
$F(u_n)$	Set of nodes in social network of user u_n .
$w^{[t]}_n$	Update pattern of user u_n during interval t (binary). $\mathbf{w}^{[t]}_n$ the update traffic volume.
$r^{[t]}_n$	Read pattern of user u_n during interval t (binary).
$t^{[t]}_{n,k}$	Schedule of u_n's updates to site S_k during interval t. $\mathbf{t}^{[t]}_{n,k}$ the update traffic volume.
$v^{\downarrow[t]}_k$	Incoming traffic volume at site S_k
$v^{\uparrow[t]}_k$	Outgoing traffic volume at site S_k
p_k	Pricing function at site S_k

Table 1: Notation for our formulation

Assuming, for the sake of simplicity, a pricing scheme based on the maximum utilization of the link (instead of the 95th percentile), it can be shown that the decision versions of optimization problem is NP-Complete. We can do a simple reduction from the *partition* problem to show this [12].

3.2 Heuristic

To keep TailGate simple, we resort to a greedy heuristic to schedule content. At a high level, we consider load on different links to be divided into discrete time bins (for instance, 1 min bins). Then, the heuristic is simple – given an upload (triggered by a write) at a given time at a given site that needs to be distributed to different sites, find or *estimate* the bin in the future in which this content will likely be read, and then 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, pick a bin at random to schedule. Simultaneous uploads are handled randomly; no special preference is given to one upload over another.

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 for upload bandwidth only; we tried a greedy variant where we optimized for upload and download bandwidth, but 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 won't be 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 large the difference, the better the results. (vi) Flash crowds are, by their very nature, unpredictable. TailGate performs as as well as other schemes when there is a flash-crowd. However, as TailGate predominantly deals with UGC (being long-tailed content), flash crowds will be rare, and in addition, UGC have very distinct (and slow) request characteristics [5].

3.3 Existing solutions

We describe two solutions – push/FIFO and a pull based approach that mimic various cache-based solutions (including CDNs) that can be used to distribute long-tailed content.

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

Immediate Push/FIFO: The content is distributed to different PoPs as soon as it is uploaded. Assuming there are no losses in the network, FIFO decreases latency for accesses as content will always be served from the nearest PoP.

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.

4. DATA DETAILS

As TailGate uses social information, the obvious questions to ask are (i) what type of information is useful and available, (ii) how can such information be used? For answering these questions, we rely on data from Twitter.

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 [20]. For these users, we then collected location information by conducting our own crawl, processed the data for junk, ambiguous information and translated everything to latitude/longitude using Google Maps API. In the end, we extracted location for 8,092,624 users from about 11M users that have actually entered location information. We use this social graph, nodes and edges only between these

nodes for our analysis in this paper. With regards to the location of the users in the dataset, we find that US has the maximum number of users (55.7%), followed by UK (7.02%) and Canada (3.9%). In terms of cities, New York has the highest number of users (2.9%), followed by London (1.7%) and LA (1.47%).

4.1 Upload Activity

For the users who have locations, we collected their tweets. Twitter allows collection of the last 3200 tweets per user, but in our dataset 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, till Nov. 2010.

This dataset is valuable in characterizing activity patterns of users. From these tweets, we extracted those tweets that contain hyperlinks pertaining to pictures (plixi, Twitpic etc.) and videos (Youtube, Dailymotion, etc.) which we consider as UGC, that resulted in 101,079,568 links. We focus our analysis on two time periods extracted from this long trace. The first one we call **day** is the set of activities on 20th May, 2010 the day we noted the maximum number of tweets in our dataset and the second one we call **week** consists of a week of activity extracted from 15th Mar, 2010 to 21st March, 2010 that is a generic week.

We recorded the size of each piece of content that is shared, resolving URL shorteners as the case may be. 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)) and 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.

4.2 Geo-distributed PoPs

To study the effects of geo-diversity on bandwidth costs, we use location data and assign users to PoPs distributed around the world. We assume the distributed architecture described in Sec. 2.1 and assume there exist datacenters in these four locations: Boston, London, LA and Tokyo[1]. We choose these locations to cover the globe. We assign users to locations using a simple method: compute the distance of a user to a location, and assign the user to the nearest location. For computing the distances we use the Haversine distance [31]. For the four locations, we get the following distribution of users: (Boston: 3,476,676, London: 1,684,101, LA: 2,045,274, Tokyo: 886,573). The east-coast of US dominates in our datasets. The relatively low number of users in Asia is because most users in Asia prefer a local version of an OSN. However, we choose Tokyo precisely for this point – users in Asia comprise social contacts of users from around the world, sharing and requesting content, adding to bandwidth costs. We find that on average, a user has 19.72 followers in her own cluster and 8.91 followers in each of the other clusters. It is well known that contacts or "friends" in social networks are located close together with respect to geographical distance [26, 22].

[1]Note that datacenter operators such as Equinix [34] already have data centers in several of these locations

4.3 Read Activity

TailGate relies on information about accesses; reads. The ideal information will be *who* requests the content, and *when*. We could not obtain direct read patterns from Twitter/FB as they are not available. So we proceed as follows.

To get an idea on *who* requests, we collected packet traces via TCPDump from an outgoing link connecting a university in northern Italy (9th Mar, 2011) to the rest of the Internet. The collection was carried out for two distinct periods: from 11H to 16H, the second from 10H to 18H. We extracted `HTTP-POST, HTTP-GET` requests corresponding to the Facebook domain. We further collected all links corresponding to pictures and videos that show up on users' wall feed. The clicks on these icons lead to new `HTTP-GET` requests, which 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. We use this as the probability of a friend requesting content (p) for our evaluation, when needed.

Note that future reads are impossible to know, but due to strong regularity in user behavior [13], an OSN like 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 others [3, 30]. In order to generate read patterns at a fine time scale (seconds), we use SONG [8], which we also describe in the Appendix.

We are interested in evaluating different scenarios that Tailgate can be effective under, as well as study 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 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 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 like 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. 2, we plot the update patterns given by the data and synthetic reads generated by us 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.

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 that context. An ideal dataset for evaluation would be the UGC that is uploaded and shared on an OSN like FB, but such data is not available to us. Instead, we use the links collected above as proxy for the media that can be shared over social networks, while recognizing that activity patterns with regards to posting and

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

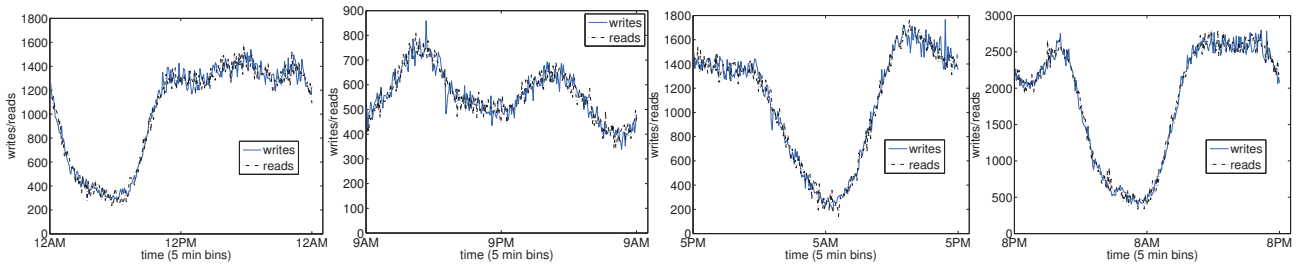

Figure 2: Writes along with synthetic reads (a) London (b) Tokyo (c) LA (d) Boston

sharing media (and large) content should follow similar diurnal patterns [30]. We assume read patterns follow a diurnal trend similar to write patterns [3, 30]. Note that we are more interested in time-of-day effects; more sophisticated read patterns where we assume content interest, quality of content etc. can be used but as we do not have sufficient information to calibrate these effects, we leave that for future work.

5. EVALUATION

In this section, we evaluate TailGate using the datasets of **day** and **week** described in Sec. 4.1 and the setup described in Sec. 4.2. We evaluate against existing schemes, namely Push and Pull (Sec. 3.3) under different models and compare bandwidth costs and the penalty metric.

5.1 Data driven evaluation

5.1.1 Metrics

The main metric we use for comparison is the 95^{th} percentile of traffic time series for the given period under study. For the day and the week dataset, we calculate the 95th percentile over 5 min bins. Reducing the 95^{th} percentile reduces bandwidth costs. We also look at penalties(P from Sec. 3), which is the number of requests pulled from the source, and indicates higher latencies for users.

5.1.2 Scenarios Examined

We describe the scenarios we study that are designed to explore the impact of different types of information on performance metrics, and to study where the benefits, if any, come from. The key inputs are the knowledge of reads and where do these reads come from. Based on these two inputs, we have the following scenarios:

Full Information, Perfect reads (FI, PR): In this model, we assume TailGate has access to the social 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 also 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. 4.3. If we assume the probability of a friend of a user requesting the content is p, then at a site the probability of requesting the content as a function of friends is $1-(1-p)^{F(u,k)}$ where $F(u,k)$ is the number of friends of user u at site k. We use $p = 0.1$ from Sec. 4.3. In order to calculate penalties, we first note that for FIFO and TailGate there are no penalties, while for PULL, penalties is simply the number of uploads that get transmitted to different PoPs, as all of these uploads will 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. 4.3. As generic reads are based on a probability distribution which in turn is based on the time of the day, we carry out simulations by assigning a success probability for writes drawn from this distribution using inverse sampling [6]. We note here that this is *conservative* and stacking 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 request happens – in effect leading to a Pull request to get data to serve that read request (Sec. 2.1). In other words, 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 (as given above). For all transmissions scheduled after the actual read request, there is a penalty, and we log the 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 will be distributed to all PoPs; bandwidth costs will consequently increase, since we pay the price of using limited and inaccurate read information.

No Information, Perfect reads (NI, PR): Clearly, this case cannot happen.

5.1.3 Results

We assume the links are empty before we start the simulations. The results of our trace driven simulations are presented in Table 2. We report the 95th percentile of the uplink bandwidth. 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 servcie providers like Google, Facebook, Yahoo etc. and their geo-diverse datacenter system). The results for this and our `day` dataset are qualitatively similar. The results are presented

Week, All (FI, PR)			
POPs	PUSH	PULL	TailGate
Boston	803.83	228.6(4.38)	164.26(4.41)
LA	562.87	187.32(1.2)	142.50(5.0)
Lon	637.85	176.78(5,66)	134(2.28)
Tokyo	273.41	95.8(3.04)	72.94(0.38)
(FI, IR)			
Boston	803.83		257.22(6.8)
LA	562.87		181.18(1.8)
Lon	637.85		210(12.9)
Tokyo	273.41		94.3(2.64)
(NI, IR)			
Boston	985.47		372.65(18.33)
LA	655.7		249.72(0.65)
Lon	850.98		354.08(0.7)
Tokyo	337.90		133.33(6.6)
Week, LT (FI, PR)			
POPs	PUSH	PULL	TailGate
Boston	419.62	70.63(2.11)	49.46(1.07)
LA	282.04	65.95(1.3)	50.83(0.46)
Lon	361.64	51.53(1.53)	39.54(0.22)
Tokyo	114.8	36.36(1.05)	32.07(0.93)
(FI, IR)			
Boston	419.62		78.17(1.39)
LA	282.04		60.29(2.4)
Lon	361.64		55.02(0.75)
Tokyo	114.8		37.8(1.45)
(NI, IR)			
Boston	498.7		119.55(4.05)
LA	324.5		97.17(2.3)
Lon	476.7		77.40(0.5)
Tokyo	141.07		38.31(1.96)

Table 2: 95th Percentiles (in MB) for all PoPs, first for all, then only long-tailed (LT). Values in brackets correspond to standard deviations

first, for all content and then only for long-tailed content (defined as objects with views < 1100). As PULL and TailGate rely on synthetic reads, we simulate multiple instances and report means, along with the standard deviations. We make the following observations:

Performance of TailGate Tailgate always outperforms PUSH in terms of decrease in the 95th percentile across all PoPs in all scenarios. When we consider the case of full information and perfect reads; TailGate reduces the 95th percentile of PUSH upto 75% and upto 28% of PULL. For specifically long-tailed (LT) content, this number goes up – upto 88% decrease over PUSH and 30% decrease over 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 (the parameter p in Sec. 4.3). When we look at the former, we note that while TailGate still performs better than PUSH (when we consider (FI,IR)), 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 126846, while for TailGate under (FI,IR), it is 12044. In other words, while the effect of inaccurate reads is comparable bandwidth costs, the penalties is still 10 times lower than PULL. Similar results for LT datasets (PULL: 95087 vs TailGate: 9162). Taken together with the conservative nature of the evaluation (Sec.5.1.2), we believe TailGate is highly competitive to PULL, in terms of bandwidth costs and increased QoE.

When we consider the inaccuracy in parameter p, we see that we have to get to $p = 0.4$ to surpass PULL in terms of bandwidth costs. In other words, TailGate seems to handle the inaccuracy in the click-rates much better than inaccuracy in read patterns.

Where do improvements come from? Under the scenario we have (4 PoPs around the world), we find that knowledge of the social graph provides modest improvement – when we consider (FI,IR) and (NI,IR), 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 as opposed to only PoPs that have friends. However, this will change if we consider more PoPs. If we consider the accuracy of reads, they seem to play a stronger role in reducing costs.

5.2 Case Study: Long-tailed videos on Youtube

In this section, we study the case where TailGate has *little* access to social information (NIIR), but can help with QoE in the case of long-tailed Youtube videos [15, 32, 37]. The entity controlling TailGate (eg. CDN) can rely on publicly available information (; 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 deploy it on four PlanetLab nodes at the same four locations – Boston, London, LA and Tokyo.

We proceed as follows: we rely on the dataset we describe in Sec. 4.2 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 the dataset described in Sec. 4.1, along with the times they were posted. Note that this information is public – anyone can collect this information. We then provide this set of writes as input to TailGate, assuming no social information (; graph structure not used) and assuming the expected reads in various locations follow a diurnal pattern. We get a schedule as output of TailGate, that 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, in effect "emulating" transfers.

After that 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 gets requested twice – first time for emulating the transfer using a schedule given by TailGate, and the second time, 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 avoid caching effects as much as possible.

We focus on the Quality of experience (QoE) for the end-user. In order to measure that, 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 if the download rate for a file drops by 70% of what was 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 time it takes for the first 15% of a file to be downloaded. As we download

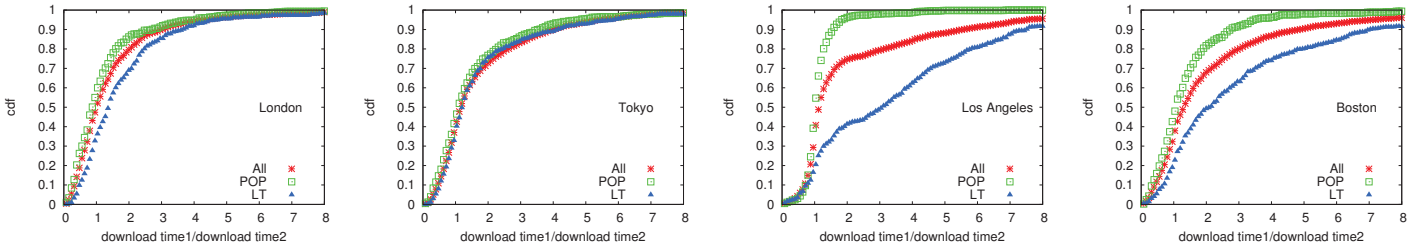

Figure 3: Performance figures for Youtube videos, improvements for download times for buffering stage

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. 3. 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" which stands for long-tailed videos ($\leq$ 1100 views). First, we note that there is an improvement of a factor of 2 and 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. 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 away (Korea) in the first place.

If we consider the results in this section taken together with reduction in bandwidth costs as reported in Sec. 5.1.3, we can conclude that a lightweight solution like TailGate can deliver long-tailed content more efficiently, while increasing performance for the end-user.

6. DEPLOYMENT SCENARIOS

OSN running TailGate: An OSN provider like 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 have considered throughout is different from that employed currently by Facebook that operates three datacenters, two on the west coast (CA) and one on the eastern side (VA) and leases space at other centers [19]. The VA datacenter operates as a slave to the CA datacenters and handles traffic from east coast US as well as Europe. All writes are handled by a datacenter in CA. However, we believe that large OSNs will eventually gravitate to the distributed architecture we described in Sec. 2.1, for the 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 [36], pointing to an architecture that relies on distributed state. If the OSN provider leases bandwidth from external providers, Tailgate decreases costs. If the provider owns the links, then Tailgate makes optimal use of the link capacity – delaying equipment upgrades as networks are normally provisioned for the peak.

CDNs with social information: Systems like CDNs are in general highly distributed (for instance Akamai), but the architecture we used in this paper captures fundamental characteristics like users being served out of the nearest PoP [16]. Existing CDN providers may not get access to social information, yet may be used by existing OSN providers to handle content (this is changing [18]). We have shown that even with limited access, the CDN provider can still optimize for bandwidth costs after making assumptions about the access patterns.

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

7. RELATED WORK

Distribution of long-tailed content has been addressed by several works, but most of the work has been confined to distribution of such content on P2P networks [25, 24]. 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. TailGate is, in part, motivated by findings presented by Wittie et al [36] where the authors analyze the current Facebook architecture and uncover network performance problems the architecture introduces, including high bandwidth utilization and large delays. The notion of distributing state to improve performance (latency) based on geography or via clustering users on a social graph has been explored by others as well [26, 17]. Tailgate can be seen complimentary to these solutions as the underlying goals – reduce bandwidth costs, and reduce end-user latency are similar.

A related work is Buzztraq that proposes to use 'social-cascades' to trigger content distribution [28]. TailGate is similar to Buzztraq, in that it relies on social information, however, TailGate does not need 'cascades' to occur to inform content distribution. In that sense TailGate is much simpler and yet effective. There has been recent work that combines information from OSNs to improve CDN service [29], and hence is similar in motivation with TailGate. The authors propose a similar mechanism to Buzztraq wherein social cascades can be used to place content close to users. TailGate also aims to place content close to users, however our focus is *when* to distribute such content to minimize bandwidth costs. TailGate can be used along with the approach proposed in [29] that answers *where*. Work by Laoutaris et al [21] describes a system called NetStitcher that aims to do bulk transfers between datacenters, by exploiting off-peak hours and storage in the network to send bulk data. NetSticher operates at the network layer, and TailGate relies on information about diurnal access patterns at the application layer. Hence TailGate and NetSticher are complimentary.

8. CONCLUSIONS

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

In this paper, we propose a lightweight solution called TailGate that exploits information available from social networks like the social graph, and regularity of activity patterns to distribute long-tailed content while decreasing bandwidth costs. Using large traces gleaned from an OSN, we show 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 Tailgate can help reduce end-user latency for long-tailed content. Taken together with bandwidth cost savings, and the fact that TailGate is a very lightweight and simple system, we envision TailGate as complementary to existing CDN technologies.

9. ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their helpful comments. We would also like to thank Josep Pujol for giving valuable feedback on an early draft. This work and its dissemination efforts have been supported in part by the FP7 project ENVISION of the European Union.

10. REFERENCES

[1] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, and A. Wolman. Volley: Automated Data Placement for Geo-Distributed Cloud Services. In *NSDI*, 2010.

[2] B. Ager, F. Schneider, J. Kim, and A. Feldmann. Revisiting Cacheability in Times of User Generated Content. In *Global Internet*, 2010.

[3] F. Benevenuto, T. Rodrigues, M. Cha, and V. Almeida. Characterizing User Behavior in Online Social Networks. In *IMC*, 2009.

[4] Burstable Billing. http://en.wikipedia.org/wiki/Burstable_billing.

[5] R. Crane and D. Sornette. Robust dynamic classes revealed by measuring the response function of a social system. *Proceedings of the National Academy of Sciences*, 105(41):15649–15653, 2008.

[6] L. Devroye. Non-uniform random variate generation, 1986.

[7] R. I. M. Dunbar. Neocortex Size as a Constraint on Group Size in Primates. *Journal of Human Evolution*, 22(6):469–493, 1992.

[8] V. Erramilli, X. Yang, and P. Rodriguez. Explore what-if scenarios with SONG: Social Network Write Generator. http://arxiv.org/abs/1102.0699, 2011.

[9] Facebook. Facebook Ranked Second Largest Video Site. http://vator.tv/news/2010-09-30-facebook-ranked-second-largest-video-site.

[10] Facebook. Facebook User Statistics. http://www.facebook.com/press/info.php?statistics.

[11] Forrester Consulting. The Future of Data Center Wide Area Networking. http://www.infineta.com/news/news_releases/press_release:5585,15851,446.

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[13] S. A. Golder, D. M. Wilkinson, and B. A. Huberman. Rhythms of Social Interaction: Messaging Within a Massive Online Network. In *C&T*, 2007.

[14] J. Hamilton. Inter-Datacenter Replication and Geo-Redundancy. http://perspectives.mvdirona.com/2010/05/10/InterDatacenterReplicationGeoRedundancy.aspx.

[15] highscalability.com. YouTube Architecture. http://highscalability.com/youtube-architecture.

[16] C. Huang, A. Wang, J. Li, and K. W. Ross. Measuring and Evaluating Large-Scale CDNs. In *IMC*, 2008.

[17] T. Karagiannis, C. Gkantsidis, D. Narayanan, and A. Rowstron. Hermes: Clustering Users in Large-Scale E-mail services. In *SoCC*, 2010.

[18] N. Kennedy. Facebook's Photo Storage Rewrite. http://www.niallkennedy.com/blog/2009/04/facebook-haystack.html.

[19] D. Knowledge. Facebook data center faq. http://www.datacenterknowledge.com/the-facebook-data-center-faq/.

[20] H. Kwak, C. Lee, H. Park, and S. Moon. What is Twitter, a Social Network or a News Media? In *WWW*, 2010.

[21] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez. Inter-Datacenter Bulk Transfers with NetStitcher. In *SIGCOMM*, 2011.

[22] D. Liben-Nowell, J. Novak, R. Kumar, P. Raghavan, and A. Tomkins. Geographic Routing in Social Networks. *Proceedings of the National Academy of Sciences*, 102:11623–11628, 2005.

[23] G. Linden. Marissa Mayer at Web 2.0. http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html.

[24] D. S. Menasche, A. A. Rocha, B. Li, D. Towsley, and A. Venkataramani. Content Availability and Bundling in Swarming Systems. In *CoNEXT*, 2009.

[25] R. S. Peterson and E. G. Sirer. AntFarm: Efficient Content Distribution with Managed Swarms. In *NSDI*, 2009.

[26] J. M. Pujol, V. Erramilli, and P. Rodriguez. Divide and Conquer: Partitioning Online Social Networks. http://arxiv.org/abs/0905.4918, 2009.

[27] 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.

[28] N. Sastry, E. Yoneki, and J. Crowcroft. Buzztraq: Predicting Geographical Access Patterns of Social Cascades Using Social Networks. In *SNS*, 2009.

[29] S. Scellato, C. Mascolo, M. Musolesi, and J. Crowcroft. Track Globally, Deliver Locally: Improving Content Delivery Networks by Tracking Geographic Social Cascades. In *WWW*, 2011.

[30] F. Schneider, A. Feldmann, B. Krishnamurthy, and W. Willinger. Understanding Online Social Network Usage from a Network Perspective. In *IMC*, 2009.

[31] R. W. Sinnott. Virtues of the Haversine. *Sky and Telescope*, 68:159, 1984.

[32] R. Torres, A. Finamore, J. Kim, M. Mellia, M. M. Munafo, and S. Rao. Dissecting Video Server Selection Strategies in the YouTube CDN. Technical Report TR-ECE-11-02, Purdue University, 2011.

[33] Twitter. Growing Around the World. http://blog.twitter.com/2010/04/growing-around-world.html.

[34] Equinix. http://www.equinix.com/.

[35] Facebook Hosts More Photos than Flickr and Photobucket. http://www.tothepc.com/archives/facebook-hosts-more-photos-than-flickr-photobucket/.

[36] 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.

[37] YouTube CDN Architecture. Private Communication, Content Delivery Platform, Google.

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

$$X(t) = m_t + \sqrt{am_t}W_t \tag{6}$$

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 'peakedness' (with the units:reads-secs) that accounts for magnitude of fluctuations.

The main method for generating synthetic reads is as follows: first generate m_t using Fourier series by first extracting the largest Fourier coefficients of the write time-series, then 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 used the top 5 Fourier coefficients to generate the diurnal pattern and used WGN with an appropriate scale parameter to generate reads. The generated time series contains the number of reads in a given time bin.