Timelines are Publisher-Driven Caches: Analyzing and Shaping Timeline Networks

Alexandre Reiffers-Masson, Eduardo Hargreaves,
Eitan Altman, Wouter Caarls, Daniel S. Menasché

ABSTRACT

Cache networks are one of the building blocks of information centric networks (ICNs). Most of the recent work on cache networks has focused on networks of request driven caches, which are populated based on users requests for content generated by publishers. However, user generated content still poses the most pressing challenges. For such content timelines are the *de facto* sharing solution.

In this paper, we establish a connection between timelines and publisher-driven caches. We propose simple models and metrics to analyze publisher-driven caches, allowing for variable-sized objects. Then, we design two efficient algorithms for timeline workload shaping, leveraging admission and price control in order, for instance, to aid service providers to attain prescribed service level agreements.

1. INTRODUCTION

Long before the creation of the world wide web, user generated content, such as personal pictures and news, was already shared through message boards over bulletin board systems (BBSes). After 1996, with the rise of the WWW, message boards of BBSes have been replaced by Internet forums, which were the precursors of news feeds and timelines as implemented by Twitter and Facebook. In a social network, each user or group of users can manage a timeline, which will be populated by user-generated content and accessed by those interested in items posted at that timeline.

Users are divided into publishers and clients. Users who generate content are referred to as *publishers*, and consumers are referred to as *clients*. While managing a timeline, a user or group of users subscribe to publishers and a subset of the contents from these publishers will be posted in the timeline. Note that in a network where most of the content is user-generated, the role of each user dynamically changes between that of a publisher and a client. *In this work, our main goal is to analyze and improve caching strategies applicable to user-generated content.*

User-generated content is fundamentally different from content generated by big providers in a number of ways. For the purposes of this work, we stress two of the fundamental differences. First, a significant portion of the user-generated content will rapidly become outdated, whereas content generated by big providers tend to be of interest during longer periods of time. In part, this is because the filtering process which occurs before the publication of user-generated content is either non-existing or much less costly than that faced by big providers before making a film or an album available to the general public. Second, when clients search for user-generated content they are typically interested in a class of items related to a given category or user. For instance, a college student might query for the last posts of his roommate during vacations. The search for content offered by big providers, in contrast, usually targets a particular item.

For the reasons discussed above, content generated by big providers and user-generated content ask for different caching policies, which turn out to coexist in today's Internet. Classical caches serve well the purposes of aiding in the provisioning of big catalogs of content, with requests issued towards specific items. Request-driven admission and eviction policies determine which contents must be added and removed from the cache *after a miss occurs*, and can be adjusted to maximize the cache utility as defined by the service provider [5, 13].

Despite their broad applicability, request-driven admission and eviction policies fall short to serve user-generated content. In essence, this is a consequence of the differences between these two kinds of contents, as described above: most of the user-generated content is short lived, and requests arrive for content classes rather than content items. Hence, we posit that publisher-driven caches are the *de facto* choice to serve such content. A publisher-driven admission and eviction policy determines which contents must be added and removed *after a content is published.*

In the simplest scenario, the analysis of publisher-driven caches can be made completely oblivious to clients interests and request rates. In this case, the occupation of timelines can be determined exclusively based on the rate at which publishers post items and the admission and eviction policies adopted by the timeline managers. First-in first-out (FIFO) placement is a natural choice; *the simplest timelines are FIFO publisher-driven caches.*

One of our aims is to devise admission strategies to maximize service providers utility. The admission strategy of items is a crucial aspect of publisher-driven caches, at the core of services like Facebook. To appreciate that point, we resort to a Twitter anecdote. In September of 2014, the CEO of Twitter announced that it was going to start filtering tweets to improve users experience [10]. However, after numerous complaints the company reported that the non-filtered version of the timeline will continue to be available [8].

To shape timelines, we consider a pricing scheme which accounts for clients interests and publishers budget. The occupation of a publisher-driven cache may be set by the service provider so as to address the possible tension between publishers who wish to have their contents in cache, and who pay for the lease of space in timelines, and users who target items of certain content classes. Client satisfaction will be a function of the probability that a user interested in a given class finds at least one item from that class in the timeline. Publishers utility is a function of the fraction of time in which its contents are made available in the timeline.

In summary, we provide the following contributions:

Timelines as publisher-driven caches: we establish a link between timelines and publisher-driven caches, and make a case for their applicability in order to server user-generated content;

Models and metrics: we propose simple models and metrics to analyze publisher-driven caches, allowing for variable-sized objects;

Workload shaping algorithms: leveraging the proposed models and metrics, we design two efficient algorithms for timeline workload shaping. The first is an admission control algorithm, posed as the solution of a geometric programming problem, and the second is a pricing control strategy which modulates the publishers publication rates so as meet a given target occupation measure at the caches.

Related work. There is a vast and growing literature on how to partition [15], prefetch [18, 6] and replicate [9] data among clusters to support the unprecedented increase in use of online social networks. To the best of our knowledge, our work is the first to consider the specifics of insertion and eviction policies of publisher-driven caches for user-generated content, and establishing its connections to timelines. An analytical study of timelines was first presented in [1], focusing on the competition between publishers for visibility [12], rather than on the dynamics and shaping of user-generated content.

Workload characterization has been considered in [17], where the authors propose a two-level shot noise model to capture the distribution of requests for contents which are clustered into categories. In [7], the authors extend the work in [17] and propose a model to account for dynamic catalogs and content popularities. In this work, in contrast, we focus on user-generated content, and assume that exogenous arrivals correspond to new opportunities to store original items. Therefore, we intrinsically account for continuously growing catalogs, wherein old items cannot be retrieved, noting that user-generated content might populate request-driven caches in case they turn out to be of interest over longer horizons.

Workload shaping for request-driven caches is recently receiving significant attention, to satisfy different service level agreements under ICN architectures [11, 13]. While previous works considered gradient descent algorithms to achieve the desired targets, our shaping algorithms consider publisher-driven caches and are inspired by previous work on stochastic approximation [2, 4].

2. MODELING TIMELINES WITH VARIABLE-SIZED OBJECTS

We start by considering a model to capture the distribution of items from different publishers in a given timeline of size K. We consider a set $\mathcal{J} := \{1, \dots, J\}$ of J publishers in a social network. Original contents from publisher j arrive at the timeline according to a Poisson process with rate λ_j, $j = 1, \dots, J$. We can easily extend the results obtained in this paper for the case of multiple timelines by defining λ_j^i as the flow of contents sent by publisher j to timeline i. Let Λ be the total arrival rate, $\Lambda = \sum_{j=1}^{J} \lambda_j$. Objects are divided into blocks of fixed size, and the timeline is divided into slots. The size of a block is assumed to be equal to that of a slot. The topmost position of the timeline is position K, and the bottommost is position 1. For convenience, if there are no objects of a given publisher in a timeline, we will refer to the position of the topmost object of such publisher as 0.

Except otherwise noted, we assume that objects may be partially stored in a timeline. In that case, and assuming that all content is admitted, upon an arrival a piece of stored content must be evicted. Depending on the size of the new object, multiple pieces might need to be removed from the timeline. Henceforth, we focus on FIFO publisher-driven caches, which means that the bottommost part of the last object to be inserted will be the first to be evicted.

Note that we may have multiple objects from the same source in a timeline at a given point in time. Let k_j be the position of the topmost content from publisher j in the timeline. The position of an object is defined as the slot occupied by the topmost element of that object. As we consider only arrivals of original content, in publisher-driven caches we do not need to track if a certain content is already at cache upon its arrival. This is in contrast to request-based caches, where placement mechanisms have to prevent duplicate objects.

Assume that upon the arrival of a content from publisher j there are no other objects from j stored at the timeline. In this case, we assume that the content from j must be stored, and we refer to the time until there are again no objects from j at the timeline as the *busy period* associated to publisher j. The busy period plays a role in publisher-driven caches similar to the characteristic-time in request-driven caches [13]. In what follows, we provide a characterization of the busy period and related publisher-driven metrics.

Let $T_j(k) \in [0, +\infty)$ be the expected period of time between the instant at which the topmost object from publisher j is at position $k \in \{0, \dots, K\}$ until the instant when the timeline does not have any objects from j. For each j, we assume $T_j(0) = 0$. We refer to $T_j(k)$ as the modified busy period, and to $T_j(K)$ simply as busy period.

Let $\mathcal{L}$ be the finite set of possible object sizes and let $p_{jl} \in [0, 1]$ be the probability that the next arrival of an object from publisher j has size equal to $l \in \mathcal{L}$, where $\sum_{l \in \mathcal{L}} p_{jl} = 1$.

Let $\mathbf{T}_j$ be the busy period vector, $(T_j(0), \dots, T_j(K))$, and let $\mathbf{A}_j$ be a $(K+1) \times (K+1)$ infinitesimal generator matrix whose kk'-entry is given by

$$\mathbf{A}_j(k, k') = \begin{cases} \lambda_j/\Lambda, & \text{if } k' = K \\ \sum_{i \in \mathcal{J} - \{j\}} p_{il} \lambda_i / \Lambda, & \text{if } k' = k - l > 0 \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

We denote by $\mathbf{I}$ and $\mathbf{1}$ the identity matrix and a vector of all ones, with dimensions $(K+1) \times (K+1)$ and $K+1$, respectively. Next, we provide a closed form expression for $\mathbf{T}_j$.

LEMMA 1. *The busy period vector is given by*

$$\mathbf{T}_j = (\mathbf{I} - \mathbf{A}_j)^{-1} \Lambda^{-1} \mathbf{1}. \quad (2)$$

Due to space limitations, all proofs are made available at [16].The busy period characterization in Lemma 1 will allow us to prove some of the results in Section 3 about the convergence of the proposed workload shaping strategies.

REMARK 2. *Alternatively, if an item is considered* off *the timeline when any part of it is cut off, the following recursive equation can be used to compute the expected busy period duration,*

$$T_j(k) = \sum_{l\in\mathcal{L}} \frac{\lambda_j p_{jl}}{\Lambda} T_j(K-l+1) + \sum_{i\in\mathcal{J}\setminus\{j\}} \sum_{l\in\mathcal{L}} \frac{\lambda_i p_{il}}{\Lambda} T_j(k-l) + \frac{1}{\Lambda}. \tag{3}$$

Next, we consider the fraction of time in which there is at least one object of publisher j at the timeline. We denote this probability by π_j, where $1-\pi_j$ is the probability that no object of publisher j is present at the timeline. The renewal theorem yields the following relationship between $1-\pi_j$ and $T_j(K)$,

$$1 - \pi_j = \frac{1/\lambda_j}{T_j(K) + 1/\lambda_j}. \tag{4}$$

Assume that for each j, $\lambda_j \in [\tilde{\lambda}_j, \hat{\lambda}_j]$, with $\tilde{\lambda}_j > 0$ and $\hat{\lambda}_j < +\infty$. Then, the characterization of $1-\pi_j$ provided in the following lemma will allow us to pose the workload shaping problem as a geometric program.

LEMMA 3. *π_j can be expressed as a posynomial.*

As an example, consider the case where all objects have size 1, i.e., $p_{j1} = 1$ for all $j \in \mathcal{L}$. In this case, the variables which characterize the content present in each position of the cache are independent and identically distributed. The probability that the timeline contains no objects of publisher j is given by a geometrically distributed random variable, $1-\pi_j = (\Lambda^{-1}\sum_{i'\neq j}\lambda_{i'})^K$.

Lemma 1 together with eq. (4) allow us to numerically compute $1-\pi_j$. Lemma (3) implies that $1-\pi_j$ is convex with respect to $\log\lambda_j$, which will be instrumental in establishing optimality guarantees of the admission control problem posed in the next section.

In this section we derived and related two timeline performance measures: the busy period and the occupation probability associated to publisher j. In the following section, we will use these results to design efficient shaping algorithms to control these two metrics.

3. WORKLOAD SHAPING

In this section our goal is to investigate how to control the flow of contents posted in a timeline. We first consider the admission control, posed as a convex optimization problem. Then, we move to pricing control mechanisms. We associate to each publisher a utility, which is a function of its corresponding busy period, and prices are used to match utilities to a set of given target values.

Admission control: a geometric programming approach. Next, we consider the admission control problem. For each $j \in \mathcal{J}$, let $\lambda_j \in [\hat{\lambda}_j, \tilde{\lambda}_j]$ be the flow rate of contents from publisher j admitted into the timeline.

The goal of the users managing a timeline is to minimize a linear combination of the probabilities that there are no messages from publisher j at the timeline, for $j = 1, \ldots, J$. Let b_j be the weight associated to publisher j. The larger the value of b_j, the more relevant are the items published by j for the users managing that timeline. Users may also wish to set a minimum level of content diversity, and to this aim we add a term to the objective function which decreases with respect to λ_j, so as to penalize small flows. Let $u_j \geq 0$ and $v_j \geq 0$ be two diversity constants (u_j is the diversity coefficient and v_j is the diversity exponent). The larger the values of u_j and v_j, the greater the penalty towards small flows from publisher j. Finally, to control the level of churn we set a constraint on the total rate of admitted contents, $\sum_{j\in\mathcal{J}}\lambda_j = \phi$, where $\phi > 0$.

In summary, the admission control problem is given as follows,

$$\min_{\boldsymbol{\lambda}} \; C(\boldsymbol{\pi}(\boldsymbol{\lambda})) = \sum_{j\in\mathcal{J}} b_j(1-\pi_j(\lambda_j)) + \gamma \sum_{j\in\mathcal{J}} u_j \lambda_j^{-v_j}, \tag{5}$$

$$\text{s.t.} \quad \sum_{j\in\mathcal{J}} \lambda_j = \phi, \tag{6}$$

$$\lambda_j \in [\hat{\lambda}_j, \tilde{\lambda}_j], \quad \forall j \in \mathcal{J}, \tag{7}$$

The parameter γ serves to balance the two terms of the objective function. It follows from Lemma 3 that the Hessian of the previous optimization problem is indefinite (the cost function is a posynomial function). For this reason, using standard algorithms such as gradient descent to solve the problem above will be difficult, as in general these algorithms will not converge, and even so, the convergence is not guaranteed to a global minimum.

To solve the admission control problem, we introduce an equivalent convex optimization problem. Let $U_j = \log(\lambda_j)$ and $U = \log(\sum_{j\in\mathcal{J}}\lambda_j)$. If we substitute λ_j by e^{U_j} in equation (4), it follows that $1-\pi_j$ is equal to $e^{-U_j}/(T_j(K) + e^{-U_j})$. Taking the logarithm of the cost function and of the constraints, the optimization problem (5) is equivalent to the following convex problem, which is in turn an instance of a geometric program [3],

$$\min_{\mathbf{U}} \; C(\mathbf{U}) = \log\left(\sum_{j\in\mathcal{J}} b_j \left(\frac{e^{-U_j}}{T_j(K) + e^{-U_j}}\right) + \gamma \sum_{j\in\mathcal{J}} u_j e^{-U_j v_j}\right), \tag{8}$$

$$\text{s.t.} \; \log\left(\sum_{j\in\mathcal{J}} e^{U_j}\right) \leq \log(\phi), \tag{9}$$

$$U_j \in [\log(\hat{\lambda}_i), \log(\tilde{\lambda}_i)], \; \forall j \in \mathcal{J}, \tag{10}$$

with $\mathbf{U} := (U_1, \ldots, U_J)$. Note that (6) has been replaced by an inequality constraint (9). Due to the fact that the cost function is decreasing in at least one component U_i for a high enough γ, the inequality constraint will always be active at the optimal admission control. Therefore, given $\pi_j(\lambda_j)$, for $j = 1, \ldots, I$, and the associated gradients, Algorithm 1 solves the admission control problem.

Pricing control: leveraging the monotonicity of the busy period. Next, our goal is to devise a pricing control scheme over publishers' flows. To this aim, we leverage two properties inherent to our model. Let q_j be the price paid by publisher j per published content. First, the flow λ_j of publisher j decreases with respect to q_j. Second, it follows from the definition and characterization of the busy

Algorithm 1: Admission control

Input: K, ϕ, $\mathbf{b}$, $\mathbf{u}$, $\mathbf{v}$, γ, δ
Output: vector of effective arrival rates $\boldsymbol{\lambda}$
Initialize: $U_j = \log(\phi)/J$ for all j
repeat
(1) compute $C(\mathbf{U})$ and $C(\mathbf{U} + \delta\mathbf{1}_j)$ using (8)
(2) $U_j \leftarrow U_j - (C(\mathbf{U} + \delta\mathbf{1}_j) - C(\mathbf{U}))/(\delta n)$, $\forall j \in \mathcal{J}$
(3) project $\mathbf{U}$ over (9) and (10) ,
until convergence.
return $\boldsymbol{\lambda} = e^{\mathbf{U}}$

Algorithm 2: Pricing mechanism

Input: K, $a \in (0,1)$, $\mathbf{q}^{(0)}$, S, $\mathcal{J}$
Output: vector of prices $\mathbf{q}$
Initialize: $q_j = q_j^{(0)}$ for all j
repeat
(1) update $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda}(\mathbf{q})$
(2) compute $T_j(K;\mathbf{q})$, $\forall j \in \mathcal{J}$
(3) update q_j using (13)
until convergence.
return $\mathbf{q}$

period that the smaller the flow of publisher i, the larger the busy period of publisher j, for $i \neq j$.

For each publisher j, we assume that its associated flow $\lambda_j(q_j) \in [\hat{\lambda}_j, \tilde{\lambda}_j]$ is a function of the price q_j (different shapes for $\lambda_j(q_j)$ are discussed and illustrated in [2]). Let us denote by $\mathbf{q} := (q_1, \ldots, q_I)$ the price vector. For each j, let $\pi_j^\star$ be the target occupancy of publisher j at the timeline. The pricing control problem is given as follows,

$$\min_{q} \quad \sum_{\forall j \in \mathcal{J}} \left(\frac{T_j(K;\mathbf{q})}{T_j(K;\mathbf{q}) + \lambda_j(q_j)} - \pi_j^\star \right)^2 \tag{11}$$

$$\lambda_j \in [\hat{\lambda}_j, \tilde{\lambda}_j], q_j \in [\hat{q}_j, \tilde{q}_j], \forall j \in \mathcal{J}. \tag{12}$$

The minimization of (11) is accomplished by adjusting the control parameter q_j as follows,

$$q_j \leftarrow \max\left(\min\left(q_j(1+a)\left(\frac{T_j(K;\mathbf{q})}{T_j(K;\mathbf{q}) + \lambda_j(q_j)} - \pi_j^\star \right), \hat{q}_j \right), \tilde{q}_j \right) \tag{13}$$

for all $j \in \mathcal{J}$. Thus, if publisher j timeline occupancy is greater than $\pi_j^\star$ its price q_j increases which produces a decrease in λ_j. Otherwise, q_j is decreased. Algorithm 2 summarizes the price adjustment strategy. Under mild assumptions, Lemma 3 together with [2] imply that Algorithm 2 converges to the optimal solution of the pricing problem.

4. CONCLUSIONS AND PERSPECTIVES

In this paper, we pose a fundamental connection between timelines used to share user-generated content and publisher-driven caches. We envision that publisher-driven and classical request-driven caches are complementary, and will remain to co-exist in future Internet architectures as two of their basic building blocks. We believe that this work opens up a number of different avenues for future work, including the control of message sizes and priorities, the estimation of users interests and its connection to the pricing mechanisms. Future work also encompasses parameterizing the proposed models using real data extracted from [14], and numerically investigating case studies with different instances of the utility functions previously analyzed under the request-driven cache setting [5, 13].

5. REFERENCES

[1] E. Altman, P. Kumar, S. Venkatramanan, and A. Kumar. Competition over timeline in social networks. In *ASONAM*. IEEE, 2013.

[2] V. Borkar and D. Manjunath. Charge-based control of diffserv-like queues. *Automatica*, 40(12), 2004.

[3] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[4] M. Crowder. Stochastic approximation: A dynamical systems viewpoint by Vivek Borkar. *International Statistical Review*, 77(2), 2009.

[5] M. Dehghan, L. Massoulie, D. Towsley, D. Menasche, and Y. Tay. A utility optimization approach to network cache design. *INFOCOM*, 2016.

[6] Salah-Eddine Elayoubi and James Roberts. Performance and cost effectiveness of caching in mobile access networks. In *Proceedings of the 2nd International Conference on Information-Centric Networking*, pages 79–88. ACM, 2015.

[7] M. Garetto, E. Leonardi, and S. Traverso. Efficient analysis of caching strategies under dynamic content popularity. In *INFOCOM*. IEEE, 2015.

[8] E. Geller. No, a 'facebook-style filter' isn't coming to twitter - yet (Washington post, September 4), 2014. `https://www.washingtonpost.com`.

[9] K. Hamidouche, W. Saad, and M. Debbah. Many-to-many matching games for proactive social-caching in wireless small cell networks. In *WiOpt*. IEEE, 2014.

[10] M. Ingram. Facebook style filtered feed is coming whether you like it or not (Gigaom, September 4), 2014. `https://gigaom.com`.

[11] S. Ioannidis and E. Yeh. Adaptive caching networks with optimality guarantees. In *SIGMETRICS*, 2016.

[12] B. Jiang, N. Hegde, L. Massoulié, and D. Towsley. How to optimally allocate your budget of attention in social networks. In *INFOCOM*. IEEE, 2013.

[13] R. Ma and D. Towsley. Cashing in on caching: On-demand contract design with linear pricing. In *CONEXT*, 2015.

[14] Giovanni Neglia, Damiano Carra, Mingdong Feng, Vaishnav Janardhan, Pietro Michiardi, and Dimitra Tsigkari. Access-time aware cache algorithms. *Inria Sophia Antipolis Technical report*, 2016.

[15] J. Pujol, V. Erramilli, G. Siganos, X. Yang, N. Laoutaris, P. Chhabra, and P. Rodriguez. The little engine (s) that could: scaling online social networks. *CCR*, 41(4), 2011.

[16] A. Reiffers, E. Hargreaves, E. Altman, W. Caarls, and D. Menasche. Timelines are publisher-driven caches: Analyzing and shaping timeline networks, 2016. `http://lia.univ-avignon.fr/chercheurs/reiffers/`.

[17] S. Traverso, M. Ahmed, M. Garetto, P. Giaccone, E. Leonardi, and S. Niccolini. Temporal locality in today's content caching: why it matters and how to model it. *CCR*, 43(5), 2013.

[18] S. Traverso, K. Huguenin, I. Trestian, V. Erramilli, N. Laoutaris, and K. Papagiannaki. Social-aware replication in geo-diverse online systems. *TPDS*, 26(2), 2015.