A Hybrid Methodology for the Performance Evaluation of Internet-scale Cache Networks

M. Tortelli[a,*], D. Rossi[a], E. Leonardi[b]

[a] *Telecom ParisTech, Paris, France*
[b] *Politecnico di Torino, Torino, Italy*

Abstract

Two concurrent factors challenge the evaluation of large-scale cache networks: complex algorithmic interactions, which are hardly represented by analytical models, and catalog/network size, which limits the scalability of event-driven simulations. To solve these limitations, we propose a new hybrid technique, that we colloquially refer to as ModelGraft, which combines elements of stochastic analysis within a simulative Monte-Carlo approach. In *ModelGraft*, large scenarios are mapped to a downscaled counterpart built upon Time-To-Live (TTL) caches, to achieve CPU and memory scalability. Additionally, a feedback loop ensures convergence to a consistent state, whose performance accurately represent those of the original system. Finally, the technique also retains simulation simplicity and flexibility, as it can be seamlessly applied to numerous forwarding, meta-caching, and replacement algorithms.

We implement and make *ModelGraft* available as an alternative simulation engine of ccnSim. Performance evaluation shows that, with respect to classic event-driven simulation, *ModelGraft* gains over *two orders of magnitude* in both CPU time and memory complexity, while limiting accuracy loss below 2%. Ultimately, *ModelGraft* pushes the boundaries of the performance evaluation well beyond the limits achieved in the current state of the art, enabling the study of Internet-scale scenarios with content catalogs comprising *hundreds billions* objects.

Keywords: Information Centric Networks; Hybrid Simulation; Caching.

1. Introduction

During the last decade, *caches* have gained momentum as the atomic part of both complex distributed networks, like Content Distribution Networks (CDNs), and emerging architectures, like Information Centric Networking (ICN) [35]. Assessing the performance of cache networks turns out to be a non-trivial task, owing to the intricate dependencies created by the interaction of several factors, such as content replacement algorithms,

*Corresponding author
Email addresses: michele.tortelli@telecom-paristech.fr (M. Tortelli), dario.rossi@telecom-paristech.fr (D. Rossi), emilio.leonardi@tlc.polito.it (E. Leonardi)

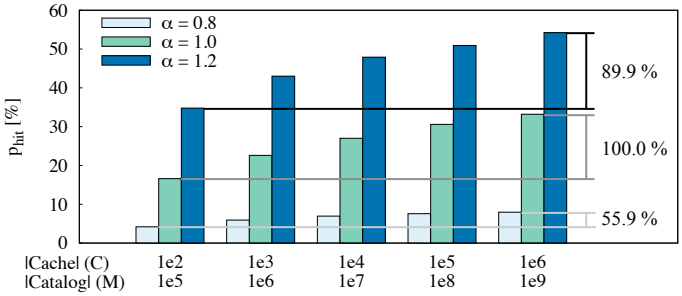


Figure 1: Hit ratio variability - 4-level binary tree, variable α, fixed $C/M = 0.01\%$ ratio, with proportional variation of C and M.

cache decision policies, forwarding strategies, network and content dynamics, etc. As a consequence, relying solely on *analytical models* [13, 23, 16, 25, 39, 12, 17, 38, 32, 29] to accurately predict network Key Performance Indicators (KPIs) becomes unfeasible. An appealing alternative, at the same time, can be that of *event-driven simulation*, considering its simplicity in describing all the algorithmic interactions between the different network entities. Nevertheless, if Internet-scale networks are targeted, CPU and memory requirements, tied to *network size* and *catalog cardinality*, would represent a difficult bottleneck to overcome. Indeed, off-the-shelf computers are not equipped with PetaBytes of RAM needed to only represent the popularity distribution of huge catalogs sized in the order of trillion object by recent estimates [33].

Common improvements often connected with simulation-based studies are *parallelization* and *downscaling*. As for the former, classical techniques [14, 10, 1] are, however, not effective in parallelizing the simulation of cache networks due to the correlation between states of neighboring caches [41]. As for the latter, instead, downscaling the scenario under investigation with naïve techniques might allow to comply with memory and CPU time constraints, on the one hand, but it can have tremendous consequences on the representativeness of the gathered results, on the other hand. For the sake of clarity, we illustrate the problem with the help of Fig. 1, which contrasts the performance of a four-level binary tree where Zipf's exponent is varied, i.e., $\alpha \in \{0.8, 1, 1.2\}$, the ratio between cache size C and catalog cardinality M is kept constant to $C/M = 0.01\%$, while both M and C are jointly downscaled. Results in Fig. 1 clearly show that barely downscaling the simulated scenario by linearly reducing the size and cardinality of all the components significantly alters system performance: the relative error between the mean hit ratio experienced in the smallest scenario and the one in the largest is between 50% and 100%. Additionally, crucial parameters like Zipf's exponent α (or Mandelbrot-Zipf plateau q) are generally measured over real Internet-scale catalogs, like YouTube [11] or BitTorrent [20], and then simply used in small-scale simulations: this clearly induces excessive distortion of the KPIs, thus invalidating the practical relevance of simulation results.

To circumvent limits of the aforementioned approaches, we propose *ModelGraft*, a hybrid methodology for the evaluation of Internet-scale cache networks that grafts ele-

ments of stochastic analysis into MonteCarlo simulations. Its design differentiates from other hybrid approaches, also extensively used in other domains than computer networks, which do not fit for the analysis of cache networks (see Sec. 9). In a nutshell, we argue that confusing the *characteristic time* T_C of Least Recently Used (LRU) caches [12, 29] with the *eviction time* of Time-To-Live (TTL) caches [23, 16] makes the respective networks practically indistinguishable. Since this equivalence constitutes the fundamental block of *ModelGraft*, and considering that the T_C approximation has been extended for many popular non-LRU caches [29] (including FIFO or Random replacement, probabilistic or complex meta-caching, dynamic arrivals, etc.), we notice that the proposed hybrid technique can be successfully applied to study complex cache networks (in terms of topologies, routing and forwarding schemes) involving a fairly general group of caching schemes for which the T_C approximation holds. We further argue that a proper *downscaling* technique can both preserve KPIs of the original Internet-scale scenario also in the downscaled TTL one (thus avoiding pitfalls shown in Fig.1), and eliminate the inefficiency of TTL caches in managing large catalogs (see Sec. 8). We finally argue that the self-contained design of *ModelGraft*, i.e., constant monitoring of KPIs and subsequent parameter correction via a feedback-loop in order to guarantee fast convergence, might increases its seamlessly adoption by end-users, since no tuning is required in practice.

In this paper, we develop the above intuitions further, making the following main contributions:

- *we propose* ModelGraft*, a novel hybrid methodology for the performance evaluation of cache networks*: its performance and accuracy are tested by means of an exhaustive simulation campaign, which reveals that CPU time and memory usage are reduced by over two orders of magnitude with respect to the classic event-driven approach, while limiting accuracy loss to less than 2%;

- *we build a fully integrated system, which we make available as open-source software*: we make the technique (and scenarios) available as open source in the latest release of ccnSim [2], where *ModelGraft* is available as an alternative simulation engine, so that users can seamlessly switch between classic event-driven vs *ModelGraft* simulations, on the same scenario (and reproduce our results).

In the remainder of this paper, of which a preliminary version appeared at [44], we first formalize the modeling background at the base of *ModelGraft* (Sec. 2). We then provide a succinct overview of our proposal (Sec. 3), followed by an in-depth description of each component (Sec. 4–6). We next validate *ModelGraft* against event-driven simulation in very-large scenarios and use it to assess system performance in Internet-scale scenarios (Sec. 7). A thorough sensitivity analysis ensures that estimated gains of *ModelGraft* are not only consistent, but also conservative, across a large set of scenario-related parameters (Sec. 8). We finally cover related work (Sec. 9) and conclude the paper (Sec. 10).

2. Stochastic modeling overview

In this section we first provide a background on Che's approximation [12] (Sec. 2.1), before analytically formalizing a fundamental building block of *ModelGraft* (Sec. 2.2), that is the equivalence between LRU caches and opportunely configured TTL caches [23].

2.1. Background

Che's[1] approximation [12], conceived for a LRU cache, is essentially a mean-field approximation which greatly simplifies the analysis of the interactions between different contents inside a cache. In particular, it consists in replacing the *cache characteristic time* $T_C(m)$ for content m (i.e., the (random) time since the last request after which object m will be evicted from the cache for the effect of the arrival of other content requests), with a *constant eviction time* T_C, which does not depend on the content itself, being a property of the whole cache. As a consequence, content m is considered to be in the cache at time t, if and only if, at least one request for it has arrived in the interval $(t-T_C, t]$. Supposing a catalog with cardinality M, for which requests are issued following an Independent Reference Model (IRM) with aggregate rate $\Lambda = \sum_m \lambda_m$, the probability $p_{in}(m)$ for content m to be in a LRU cache at time t can be expressed as:

$$p_{in}(\lambda_m, T_C) = 1 - e^{-\lambda_m T_C}. \tag{1}$$

Denoting with $\mathbb{1}_{\{\mathrm{A}\}}$ the indicator function for event A, and considering a cache of size C, we have, by construction, that $C = \sum_m \mathbb{1}_{\{m \text{ in cache at t}\}}$. Then, after averaging both sides, we obtain:

$$C = \sum_m \mathbb{E}\left[\mathbb{1}_{\{m \text{ in cache at t}\}}\right] = \sum_m p_{in}(\lambda_m, T_C). \tag{2}$$

It follows that the characteristic time T_C can be computed by numerically inverting (2), which admits a single solution [12]. It is worth to notice that this strong correlation between the cache size C and the characteristic time T_C will be the foundation of the *self-stabilization* property of *ModelGraft* (Sec. 6), where TTL caches will be used instead of their LRU counterpart (Sec. 2.2).

Finally, KPIs of the system, such as the *cache hit probability* p_{hit}, can be computed using the PASTA property:

$$p_{hit} = \mathbb{E}_\Lambda[p_{in}(\lambda_m, T_C)] = \sum_m \lambda_m p_{in}(\lambda_m, T_C) / \sum_m \lambda_m. \tag{3}$$

As far as *a single cache* is concerned, Che's approximation was originally proposed for LRU caches [12], but it has been extended in more recent times to FIFO or Random replacement [17], LRU caches with probabilistic insertion [29], possibly depending on complex cost functions [5], and renewal request models [29]. For the sake of clarity, we limit our narrative to LRU caches, despite the *ModelGraft* methodology applies to the whole class of caching policies for which a Che's approximation has been derived. For the sake of completeness, we also report experimental results of non-LRU cache networks in Sec. 7.

As far as *cache networks* are concerned, however, further approximations are required as an alternative approach to the computationally and algorithmically challenging characterization of the miss streams at any node in the network [29, 32]. Approximations that

[1] Interestingly, it was recently brought to our attention that Fagin [15] already published in 1977 the results (on "computer" caches) independently found by Che [12] in 2002 (on "network" caches).

often lead to a significant degradation of the accuracy with respect to simulation [43], especially in arbitrary networks with shortest path [39] or more complex routing policies [42], where estimation errors can potentially cascade. In addition, analytical approaches often assume stationary conditions, thus lacking in characterizing transient periods, although a model has been recently proposed only for a single cache [19]. All these reasons thus justify the quest for a hybrid approach, such as the one we propose in this work.

2.2. Equivalence of LRU and TTL caches

Observe that, under Che's approximation, i.e., given the *characteristic time* T_C (also called eviction time) the dynamics of the different contents become completely decoupled. Therefore, we can greatly simplify the analysis of complex and large cache networks by replacing every LRU cache with a simpler Time-to-Live (TTL) cache [23, 16, 30, 32], where contents are evicted upon the expiration of a pre-configured *eviction timer* T'_C, which, for each content, is set upon the arrival of the last request if the content is not in the cache, and reset at every subsequent cache hit.

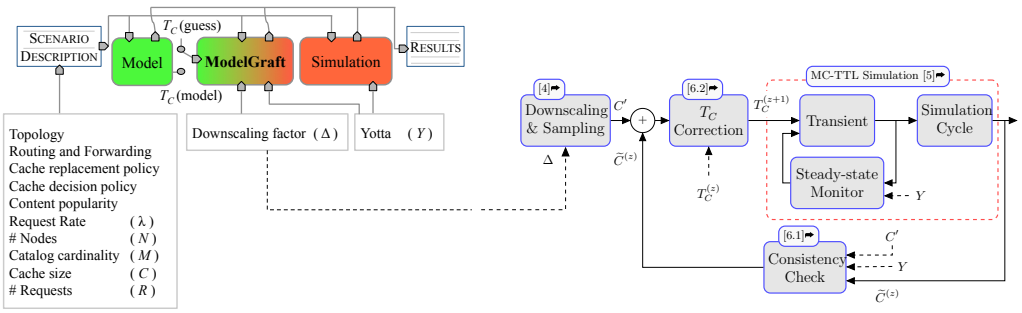
Observation 1. *As experimentally shown in [12], and remarked in [16, 29], the dynamics of a LRU cache with characteristic time T_C, fed by an IRM process with a catalog $\mathcal{M}$ of cardinality M, and request rates λ_m drawn from a distribution Λ, become indistinguishable from those of a TTL cache with deterministic eviction timer T'_C set equal to T_C (i.e., $T'_C = T_C$), and operating on the same catalog*:

$$p_{hit}^{TTL}(T'_C) = p_{hit}^{TTL}(T_C) = \mathbb{E}_\Lambda[1 - e^{-\lambda_m T_C}] = \mathbb{E}_\Lambda[p_{in}(\lambda_m, T_C)] = p_{hit}^{LRU}(T_C) \tag{4}$$

Specifically, (4) equals the average hit probability of the original LRU system to that of its TTL-based equivalent [16, 29]. We however point out that the size of a TTL cache is not bounded a priori, and it equals that of the correspondent LRU one only in expectation; since contents are evicted only after the expiration of their timer, the number of cached objects could exceed that of the original LRU system, thus translating in *even higher CPU and memory requirements*, which hampers the use of TTL caches in Internet-scale simulations. This also explains why, though intellectually interesting, TTL caches have been used, so far, only for very simple networks (e.g., single cache, trees, etc.) and catalogs (i.e., few hundreds objects) [23, 16, 30, 32]. However, the following argument makes a crucial step toward the analysis of large-scale cache networks:

Observation 2. *Large-scale LRU networks can be analyzed through a downscaled system associated to a catalog $\mathcal{M}'$ with cardinality $M' \ll M$, where each cache is replaced by its TTL equivalent with an eviction time T'_C set equal to the characteristic time T_C of the original LRU cache. KPIs of the original network are, by construction, recovered by averaging system performance over multiple MonteCarlo realizations of the downscaled one, each lasting for a duration δT, and where rates λ'_m for individual objects of the downscaled catalog $\mathcal{M}'$ are uniformly and independently drawn from Λ. Expanding (4):*

$$\begin{aligned} \mathbb{E}_\Lambda\left[p_{in}(\lambda'_m, T_C)\right] &= \frac{\mathbb{E}_\Lambda\left[\lambda'_m p_{in}(\lambda'_m, T_C)\right]}{\mathbb{E}_\Lambda[\lambda'_m]} \\ \frac{\sum_m \lambda_m P_t(\lambda'_m = \lambda_m) p_{in}(\lambda_m, T_C)}{\sum_m \lambda_m P_t(\lambda'_m = \lambda_m)} &= \frac{\sum_m \lambda_m p_{in}(\lambda_m, T_C)}{\sum_m \lambda_m} \end{aligned} \tag{5}$$



(a) Integration in ccnSim (b) Synoptic of ModelGraft Workflow

Figure 2: *ModelGraft* overview.

where the top-right expression represents the ratio between the average hit-rate and the average rate of requests at the cache, and $P_t(\lambda'_m = \lambda_m)$ represents the probability that $\lambda'_m = \lambda_m$ at a generic time instant t.

Given that contents have decoupled dynamics under the previous assumptions, we can (i) *significantly downscale the system*, thus reducing both memory and CPU time, and, in parallel, (ii) *accurately represent* complex interactions and correlations among different caches.

Observation 3. *A practical approximation is to let $\delta T \to 0$, and re-extract λ'_m at every new request, thus satisfying (5).*

We remark that, in this case, in order to ensure that at a given arbitrary time t, $P_t(\lambda'_m = \lambda_m) = 1/M$, we have to extract the new value for λ'_m at every arriving request from Λ non uniformly. In particular, it can be easily shown, as consequence of classical renewal arguments, that the probability of extracting λ_m as a new value for λ'_m must be set equal to $\frac{\lambda_m}{\sum_m \lambda_m}$, i.e., more formally, $P(\lambda'_m = \lambda_m) = \frac{\lambda_m}{\sum_m \lambda_m}$.

The remainder of this paper illustrates, describes, and validates in greater details the methodology that is built upon these observations.

3. *ModelGraft* overview

3.1. ModelGraft *components and workflow*

Before delving into *ModelGraft*'s details, we analyze it in the context of the full framework made available as open-source through the latest version of ccnSim [2] (although the methodology is portable to other simulators [43]), and we illustrate each of its building blocks at high level. As illustrated in Fig. 2-(a), starting from a unique scenario description (also available at [2]), users can analyze the performance of cache networks via either an *analytical model* [29] (when available, left), a classic *event-driven* simulation engine (right), or via the *ModelGraft* engine (middle).

ModelGraft performs MonteCarlo simulations of opportunely downscaled systems, where LRU caches are replaced by their Che's approximated version, implemented in practice as TTL caches. It formally depends on a single additional parameter, namely

Table 1: Scope of *ModelGraft* applicability

	Supported (tested)	*Supported (untested)*	Unsupported
Workload	IRM, Dynamic	-	-
Forwarding	SP, NRR[42], LoadBalance[41]	-	-
Cache decision	LCE, LCP[6], 2-LRU[29]	CoA[5], LAC[9]	LCD/MCD[25]
Cache replacement	LRU[12], FIFO[29], RND[17]	-	LFU

the *downscaling factor* Δ, which can be easily tuned according to guidelines in Sec. 6.2. As introduced in Sec. 2.2, each cache in the network should receive, as input, its characteristic time T_C. One option could be to bootstrap *ModelGraft* with *informed guesses* of T_C gathered via, e.g., analytical models (notice the $T_C(model)$ switch in Fig. 2-(a)); nevertheless, the choice of routing/forwarding strategies, catalog dynamics, and so on, would be limited, since only cases covered by the model could be considered, thus restricting the generality of the methodology itself. A more interesting approach, used by default in *ModelGraft*, is instead to start from *uninformed guesses* of T_C (notice the default wiring to the $T_C(guess)$ switch in Fig. 2-(a)), and let the system iteratively correct input values. In this case, no algorithmic restrictions are made on the system, which is simulated with a TTL-based MonteCarlo approach: this is possible since *ModelGraft* is intrinsically conceived as an auto-regulating system, so that, by design, it *converges to accurate results* even when the input T_C values (that users do not even need to be aware of) *differ by orders of magnitude* from the correct ones.

Individual building blocks of *ModelGraft* are exposed in Fig. 2-(b), which we overview here and thoroughly describe in the following sections. In a nutshell, *ModelGraft* starts with the configuration of the *downscaling and sampling* process (Sec. 4), before entering the *MonteCarlo TTL-based (MC-TTL) simulation* (Sec. 5). During the MC-TTL phase, statistics are computed after a transient period (Sec. 5.1), where an *adaptive steady-state monitor* tracks and follows the dynamics of the simulated network in order to ensure that a steady-state regime is reached without imposing a fixed threshold (e.g., number of requests, simulation time, etc.) a priori (Sec. 5.2). Once at steady-state, a downscaled number of requests are simulated within a *MC-TTL cycle* (Sec. 5.3), at the end of which the monitored metrics are provided as input to the *self-stabilization* block (Sec. 6): a *consistency check* decides whether to end the simulation (Sec. 6.1), or to go through a T_C *correction* phase (Sec. 6.2) and start a new simulation cycle.

3.2. ModelGraft *applicability*

Identifying systems that can be targeted by *ModelGraft* is an important aspect worth elucidating further. We summarize in Tab. 1 those features/algorithms implemented in ccnSim, which are either supported by *ModelGraft* or available only with the event-driven engine. Intuitively, as long as the original system admits a mean-field approximation à la Che, the dynamics of its contents can be decoupled using the characteristic time T_C, which can be in turn used as input for TTL caches. While the original Che's approximation holds only for LRU caches, as shown in [29], the class of cache networks that can be modeled by its extensions is fairly large.

In particular, Che's approximation has been already extended [17, 29] to cover most of the classical *cache replacement strategies*, such as LRU, RANDOM, and FIFO under

IRM and non-IRM traffic patterns with dynamic popularities [18], for which *ModelGraft* can be used (a theoretical justification is given in [26]). While *ModelGraft* is not directly applicable to Least Frequently Used (LFU), it can possibly work with approximations of LFU which are content-popularity unaware – at the same time, we point out that LFU is not interesting in practice as it is far from optimum in scenarios with dynamic popularity, where the supported 2-LRU and variants are preferable. In addition, *ModelGraft* supports a fair variety of *cache decision policies* and *forwarding strategies*; as for the former ones, Leave Copy Everywhere (LCE), Leave Copy Probabilistically (LCP)[6], k-LRU (whose behavior has been proven to converge to LFU [26]), and newer Cost-Aware (CoA) [5] and Latency-Aware Caching (LAC)[9] are supported (whereas LCD and MCD are untested). As for the latter ones, instead, not only Shortest Path (SP), but all state-of-the-art, including Load Balance [41] and Nearest Replica Routing (NRR)[42], are supported. Overall, Tab.1 confirms that *ModelGraft* has a rather wide spectrum of applicability, of which we provide some example in the following sections.

4. Downscaling and sampling

4.1. Design

The *ModelGraft* workflow starts with a proper *downscaling* of the original scenario. The only controlling parameter is the *downscaling factor* $\Delta \gg 1$: a catalog comprising $M' = M/\Delta$ objects, attracting $R' = R/\Delta$ total requests, will be simulated at steady-state in the equivalent TTL-based system (Sec. 5.3). Moreover, when $T'_C = T_C$, a TTL cache in the downscaled system will store, on average, $C' = C/\Delta$ contents at steady-state. Indeed, adapting (2) to the downscaled scenario (i.e., $M' = M/\Delta$), we can compute the expected cache size as:

$$C' = \sum_{n=1}^{M'} p_{in}(\lambda'_n, T_C) = C/\Delta. \tag{6}$$

However, in order to avoid pitfalls caused by a naïve downscaling process (recall Fig. 1), we need to ensure that the downscaled catalog preserves the statistical properties of the original popularity distribution (5). While our methodology is not restricted to a specific popularity law, in what follows we develop the case where object popularity follows a Zipfian probability distribution with exponent α – which is also the most interesting case from a practical viewpoint.

The proposed approach, sketched in Fig. 3, consists in splitting the original catalog into a number of M' bins having the same cardinality Δ, i.e., $|\mathcal{M}_n| = \Delta$, where $\mathcal{M}_n$ refers to the n-th bin, with $n \in [1, M']$. Each bin is represented in the downscaled system by a single "meta-content", thus obtaining an active catalog of M' meta-contents. The key idea is to let each meta-content n be requested with an *average request rate*, $\bar{\lambda}'_n$, equal to the average request rate of the original contents within the respective bin. More formally, for the n-th meta-content, with $n \in [1, M']$, it is required that:

$$\bar{\lambda}'_n = \frac{1}{\Delta} \sum_{m=(n-1)\Delta+1}^{n\Delta} \lambda_m \triangleq \mathbb{E}[\lambda_n], \tag{7}$$

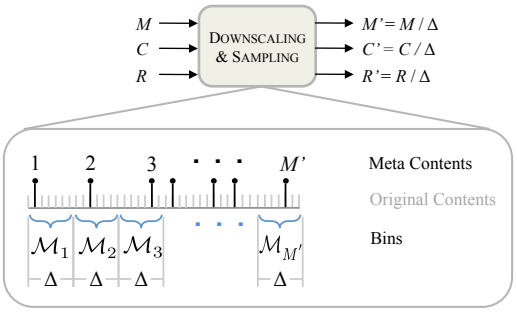


Figure 3: Downscaling and sampling process.

where the interval $[(n-1)\Delta+1, n\Delta]$ represents contents of the original catalog that fall within the n-th bin, $\lambda_m = \Lambda m^{-\alpha} / \sum_{k=1}^{M} k^{-\alpha}$ is the rate for the m-th content in the original catalog, and $\mathbb{E}[\lambda_n]$ is the average request rate for the contents within the n-th bin. The aforementioned design can be implemented by (i) instantiating M' request generators in parallel, i.e., each one $n \in [1, M']$ issuing requests for a single meta-content, and by (ii) varying their instantaneous request rate at each new request, as suggested in Observation (3), so that their average comply with (7).

4.2. Implementation

It is easy to see that the simplest implementation of the above requirements boils down to binding the sampling probability of the request rate for the n-th meta-content λ'_n at every new request, i.e., $P(\lambda'_n = \lambda_m)$, with the rate of the corresponding sampled content, i.e., the m-th one in the original catalog, among the Δ inside the respective bin, i.e., $m \in [(n-1)\Delta+1, n\Delta]$:

$$P(\lambda'_n = \lambda_m) = \frac{\lambda_m}{\sum_{j=(n-1)\Delta+1}^{n\Delta} \lambda_j} = \frac{1}{\Delta} \frac{\lambda_m}{\mathbb{E}[\lambda_n]}. \tag{8}$$

While the above requirement (7) is met, a significant downside of this naïve implementation is its memory allocation. Indeed, since it is based on the classic inverse transform sampling, this approach would require to store M' Cumulative Distribution Functions (CDFs) having each a size Δ, resulting in an overall memory allocation equal to $M'\Delta = M$ elements, i.e., as in the original scenario. Given that M is the dominant factor driving the overall memory occupancy, it is clear that such a simple implementation is not compatible with our goals.

We therefore resort to a better sampling technique called Rejection Inversion Sampling [22], which is an acceptance-rejection method that efficiently generates random variables from a monotone discrete distribution (in this case Zipf's distribution) without allocating memory-expensive CDFs, and which is characterized by a $\mathcal{O}(1)$ runtime complexity. Originally proposed in the late 90s [22] for $\alpha > 1$, this technique has been very recently extended to all non-negative exponents $\alpha > 0$ [4]. Without thoroughly discussing all its details, we provide a brief overview of its main steps. First of all,

the object population (that we assume being the interval $[1, \Delta]$ for sake of simplicity) is divided into two parts: the *head*, composed by only the first element, and the *tail*, consisting in the remaining objects. Then, as in any acceptance-rejection method, a *hat* function $h(x)$ and its *integral* $H(x)$ are defined. At this point, the algorithm [4] iterates through the following steps: (i) extract a uniformly distributed random variable U from the area under the hat function; (ii) extract, by inversion, the element X to test, and limit its range to $[1, \Delta]$, i.e.: $X \leftarrow H^{-1}(U)$, $K \leftarrow \lfloor X + 1/2 \rfloor$ (iii) return K if: $K - X \leq s \parallel U \geq H(K+1/2) - h(k)$. It can be shown [4] that the probability of selection and acceptance for both the head and an element from the tail are proportional to the probability mass function of Zipf's distribution. Recall now that power-law distributions (and hence Zipf's one) exhibit a *scale-independent* (or *self-similar*) property, according to which the scale exponent α is preserved independently of the level of observation. Hence, by means of *rejection inversion sampling*, we can consider a single interval [1,Δ] (i.e., with the same cardinality of one bin) from which extracting indexes that follow a Zipf's distribution with exponent α. In turn, this generator is used to vary the request rates of the M' meta-contents at each new request according to (8): this implies a gain of the same order of magnitude of the catalog size M with respect to a classic inversion sampling technique that stores the content popularity CDF for the whole catalog. As a consequence, the pivotal role of inversion rejection sampling in studying cache dynamics at Internet-scale (i.e., with trillion objects [33]) becomes clear. Formally (proof available in [45]):

Observation 4. *If the request generator associated to the n-th bin/meta-content, with $n \in [1, M']$, needs to schedule the next request, an index $t \in [1, \Delta]$ is extracted with the aforementioned technique; to satisfy condition (7), the correspondent request rate has to be computed as $\lambda'_n = \lambda_{(n-1)\Delta+t}$.*

5. MC-TTL Simulation

5.1. Transient

Once the scenario is properly downscaled, *ModelGraft* starts the warm-up phase of the first MC-TTL *simulation cycle* (with initial uninformed guesses for T_C). Given that the duration of the warm-up can be affected by many parameters, *ModelGraft* automatically adapts its duration by monitoring KPIs in order to guarantee their statistical relevance. For instance, the persistence of a transient period can be affected by forwarding policies (e.g., Nearest Replica Routing comes with shorter average paths, which can reduce the transient with respect to shortest path [42]), as well as by cache decision policies (e.g., like LCP [6], where the reduced content acceptance ratio with respect to LCE is expected to yield longer transients).

5.2. Steady-state monitor

The convergence of a single node i is effectively monitored using the Coefficient of Variation (CV) of the *measured hit ratio*, $\bar{p}_{hit}(i)$, computed via a *batch means* approach. In particular, denoting with $p_{hit}(j, i)$ the j-th sample of the measured hit ratio of node

i, node i is considered to enter a steady-state regime when:

$$CV_i = \sqrt{\frac{1}{W-1}\sum_{j=1}^{W}\left(p_{hit}(j,i) - \bar{p}_{hit}(i)\right)^2}\left(\frac{1}{W}\sum_{j=1}^{W}\bar{p}_{hit}(j,i)\right)^{-1} \leq \varepsilon_{CV}, \qquad (9)$$

where W is the size of the sample window, and ε_{CV} is a user-defined convergence threshold (a sensitivity analysis of ε_{CV} is extensively reported in [45] and briefly summarized in Sec.8.5). To avoid biases, new samples are collected only if (i) the cache has received a non-null number of requests since the last sample, and (ii) its state has changed, i.e., at least a new content has been admitted in the cache since the last sample[2].

At network level, denoting with N the total number of nodes in the network, and given a tunable knob $Y \in (0, 1]$, we consider the whole system to enter steady-state when:

$$CV_i \leq \varepsilon_{CV}, \qquad \forall i \in \mathcal{Y}, \qquad (10)$$

where $|\mathcal{Y}| = \lceil YN \rceil$ is the *set of the first YN nodes satisfying condition* (9). The rationale behind this choice is to avoid to unnecessarily slow down the convergence of the whole network by requiring condition (9) to be satisfied by all nodes: indeed, due to particular routing protocols and/or topologies, there are nodes that have low traffic loads (hence, long convergence time), and, at the same time, a marginal weight in network KPIs. A sensitivity analysis of the impact that Y has on system performance is reported in Sec. 8: shortly, by excluding few (slow) nodes, the accuracy of KPIs is not affected (as these nodes do not bring crucial information), despite convergence time is significantly reduced (as we exclude the slowest nodes).

5.3. Simulation cycle

For the original system, the *duration of a simulation cycle* T at steady-state is computed as $T = R/(\Lambda N_C)$, where R is the target number of requests, $\Lambda = \sum_{i \in M} \lambda_i$ is the aggregate request rate per client, and N_C the number of clients. In MC-TTL simulations, instead, the total request rate per each client is $\Lambda' = \sum_{n \in M'} \lambda'_n \approx \Lambda/\Delta$. Keeping the simulated time $T' = T$ constant, it follows that the number of simulated events per each cycle of a MC-TTL simulation is $R' = R/\Delta$ – with an expected significant reduction of the CPU time required to simulate a cycle in classic event-driven simulation.

6. Self-stabilization

From a practical viewpoint, one of the most valuable and desirable properties of our hybrid methodology is a *self-contained* design that allows to simulate large scale networks even in the absence of reliable estimates of characteristic times T_C. This is achieved through a feedback loop, which ensures that our methodology *self-stabilizes* as a result of the combined action of two elements: a measurement step, referred as *consistency check*, and a *controller action*, where inaccurate T_C values are corrected at each iteration.

[2] To exemplify why this is important, consider that with a LCP(p) cache decision policy, where new contents are probabilistically admitted in the cache, the reception of a request is correlated with the subsequent caching of the fetched content only in 1 out of $1/p$ cases.

6.1. Consistency check

The *consistency check* is based on Eq. (6), according to which a TTL cache of the downscaled system, with downscaling factor Δ, and $T'_C = T_C$, would store, on average, $C' = C/\Delta$ contents at steady-state. Considering (i) the absence of size boundaries for TTL caches [16], (ii) the expected cache size $\mathbb{E}[\widetilde{C}] \approx C' = C/\Delta$ for a TTL cache in the downscaled system, and (iii) the strong correlation that exists between the eviction time T_C and the number of cached contents, it follows that we can consider the *measured cache size* $\widetilde{C}$ as our controlled variable. In particular, for each TTL cache we maintain an online average of the number of stored contents as:

$$\widetilde{C}_i^{(z)}(k+1) = \frac{\widetilde{C}_i^{(z)}(k)\ t(k) + {B_i}^{(z)}(k+1)\ [t(k+1) - t(k)]}{t(k+1)}, \tag{11}$$

where $\widetilde{C}_i^{(z)}(k)$ is the online average of the cache size of the i-th node at k-th measurement time during z-th simulation cycle, and ${B_i}^{(z)}(k+1)$ is the actual number of contents stored inside the TTL cache of the i-th node at the $(k+1)$-th measurement time during the z-th simulation cycle. Samples for the online average are clocked with *miss* events and collected with a probability 1/10, so that samples are geometrically spaced, in order to avoid oddities linked to periodic sampling [7].

At the end of each MC-TTL simulation cycle (i.e., after the simulation of R' requests), the *consistency check* block evaluates the discrepancy of the measured cache size $\widetilde{C}^{(z)}$, with respect to the target cache C', by using the following expression:

$$\frac{1}{YN} \sum_{i \in \mathcal{Y}} \frac{\left| C' - \widetilde{C}_i^{(z)}(k_{end}) \right|}{C'} \leq \varepsilon_C, \tag{12}$$

where $\widetilde{C}_i^{(z)}(k_{end})$ is the online average of the measured cache size of i-th node at the end of the z-th simulation cycle, C' is the target cache (that, for the sake of simplicity, we consider equal for all the nodes), and ε_C is a user-defined consistency threshold. For coherence, measures are taken on those $|\mathcal{Y}| = YN$ nodes that have been marked as stable in Sec. 5.2. If condition (12) is satisfied, the MC-TTL simulation ends, otherwise a new MC-TTL cycle needs to be started: T_C values are corrected (as in the next subsection), all the caches are flushed, and the online average measurements are reset.

6.2. T_C correction

The controller action leverages the direct correlation that exists between the *target cache size* $C' = C/\Delta$ expected for a TTL cache at steady-state, and its characteristic time $T'_C = T_C$, that is expressed through equations (1)-(2)-(4)-(12). Intuitively, there exists a direct proportionality between C' and T_C: the average number of elements $\widetilde{C}$ stored in a TTL cache, with TTL=T_C, grows as T_C grows.

Therefore, if the consistency check block reveals that the measured cache size $\widetilde{C}$ of a particular node is *smaller* than its target cache, $\widetilde{C} < C'$, it means that the respective T_C value provided as input is actually *smaller* than the true T_C, thus suggesting an increment in the next step. Viceversa, for $\widetilde{C} > C'$, the T_C of the correspondent node should be decreased.

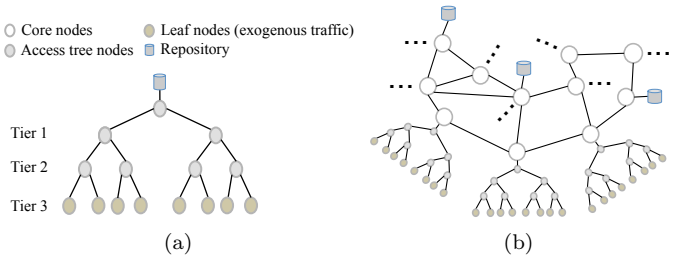


Figure 4: Network Topologies: (a) 4-level binary tree, (b) CDN-like.

As a consequence of this relationship, we employ a *proportional* (P) controller to compensate for T_C inaccuracies. That is, if condition (12) is not satisfied at the end of z-th simulation cycle, T_C values are corrected, before starting the next cycle, as:

$$T_{Ci}{}^{(z+1)} = T_{Ci}{}^{(z)} \left(\frac{C'}{\widetilde{C}_i^{(z)}} \right), \tag{13}$$

where $T_{Ci}{}^{(z)}$ is the TTL value assigned to the i-th node during the z-th simulation cycle. In practice, (13) guarantees a fast convergence towards the right T_C values (see Sec. 8 and [45] for a deeper analysis), avoiding, at the same time, any divergence of the control action (provided that measures on $\widetilde{C}$ are taken at steady-state). This ensures that *ModelGraft* would often provide considerable gains, even when inaccurate input T_C values require the execution of multiple simulation cycles.

There is an important condition worth highlighting: i.e., the controller needs to react on *reliably measurable* quantities, as opposite to *noisy measures* – which happens whenever the terms of the ratio are very small $C' \approx \widetilde{C}_i^{(z)} \approx 1$, as errors in their estimation amplify in their ratio $C'/\widetilde{C}_i^{(z)}$. In particular, this translates into a very simple practical guideline: a lower bound to the target cache size of the downscaled system can be introduced as $C' = C/\Delta \geq 10$, which practically upper bounds the maximum downscaling factor to $\Delta \leq C/10$ (see [45] for further details).

7. Results

We now validate the *ModelGraft* engine against classic event-driven simulation in very-large scale scenarios (Sec. 7.1), and we then project *ModelGraft* gains to Internet-scale ones, which are prohibitively complex for classic event-driven simulators (Sec. 7.2). To thoroughly assess *ModelGraft* gains, we test its performance not only with *accurate* T_C *values* extracted from event-driven simulations and provided as input (Sec. 7.1–7.2), but also with *rough* T_C *guesses*, possibly inaccurate by several orders of magnitude, in order to stress the self-stabilization capabilities of *ModelGraft* (see Sec. 8). All the results presented in this section have been obtained by executing both event-driven and *ModelGraft* simulations on the same commodity hardware, i.e., an Intel Xeon E5-1620, 3.60GHz, with 32GB of RAM.

Table 2: *ModelGraft* validation, accurate initial T_C (4-level binary tree, $M = R = 10^9, C = 10^6, \Delta = 10^5, Y = 0.75$)

Cache Decision Policy	Technique	p_{hit}	Loss	CPU time	Gain	Mem [MB]	Gain
LCE	Simulation	33.2%	**1.8%**	11.4 h	**194x**	6371	**168x**
	ModelGraft	**31.4%**		**211 s**		**38**	
LCP(1/10)	Simulation	35.4%	**1.4%**	7.3 h	**90x**	6404	**168x**
	ModelGraft	**34.0%**		**291 s**		**38**	
2-LRU	Simulation	37.0%	**0.9%**	10.8 h	**97x**	8894	**234x**
	ModelGraft	**36.1%**		**402 s**		**38**	

7.1. ModelGraft *validation: Very Large-scale Scenario*

To evaluate the accuracy of *ModelGraft*, we observe that simulators considered in [43] (ccnSim, NDNsim and Icarus, to mention a few) all yield consistent results: it follows that comparing *ModelGraft* against one of these simulators is sufficient to confirm *ModelGraft* validity. We thus select ccnSim –which was already shown to be among the most scalable ICN software tools [43]– and consider the largest scenario we can investigate via its event-driven engine. To stretch the boundaries reachable by event-driven simulation even further, we integrate the rejection inversion sampling in ccnSim – so that, to the best of our knowledge, *we are the first to evaluate ICN networks when the content catalog is in the order of billions.*

The validation scenario represents an ICN-access tree [31] with N=15 nodes, as depicted in Fig. 4(a). A single repository, connected to the root node, stores a $M = 10^9$ objects catalog, where objects follow a Zipf popularity distribution with exponent $\alpha = 1$. $R = 10^9$ overall requests are injected through leaf nodes, at a rate of $\Lambda = 20$ req/s per leaf, following an Independent Reference Model (IRM).

The cache size of each node is fixed at $C = 10^6$ (i.e., cache to catalog ratio of $C/M = 0.01\%$). We consider three cache decision policies: (i) LCE, where fetched contents are always cached in every traversed node; (ii) LCP(1/10), that probabilistically [6] admits contents in the cache (one over ten fetched contents is cached on average); (iii) 2-LRU [24, 29], where cache pollution is reduced by using an additional cache in front of the main one, with the purpose of caching only the names of requested contents: the fetched contents will be stored in the main cache only in case of a name hit in the first cache. According to our rule of thumb $C' = C/\Delta \geq 10$ (anticipated in Sec. 6.2 and substantiated in [45]), the maximum downscaling factor is $\Delta = 10^5$. Additionally, equations (10) and (12) are computed considering $Y = 0.75$, $\varepsilon_{CV} = 5 \cdot 10^{-3}$, and $\varepsilon_C = 0.1$: in other words, we test convergence of 75% of the caches in the network, by requiring the coefficient of variation of the hit rate to be below $5 \cdot 10^{-3}$, and we iterate *ModelGraft* simulations until the measured average cache size of those nodes is within 10% of the expected size $C' = C/\Delta$ (a sensitivity analysis of the above parameters is reported in Section.8).

Tab. 2 reports mean values (computed over 10 different runs) for three KPIs: mean hit ratio p_{hit}, CPU time, and RAM memory occupancy. Confidence intervals, instead, are reported in the sensitivity analysis section available at [45]. The table also highlights *relative gains* for CPU and memory footprint (i.e., $KPI^{Simulation}/KPI^{ModelGraft}$), as well as *accuracy loss* with respect to simulation (i.e., $|p_{hit}^{Simulation} - p_{hit}^{ModelGraft}|$). Sev-

eral observations follow from the table. First of all, (i) *ModelGraft* brings significant improvements in terms of reduction of both CPU time and memory occupancy; indeed, the relative gains with respect to the classic event-driven approach are always about two orders of magnitude large, regardless of the cache decision policy. Considering the LCE case as an example, *ModelGraft* requires only 38 MB of memory and 211 seconds of CPU time, compared to 6.4 GB and 11 hours under classic simulation. Additionally, we can also highlight the fact that, despite the aforementioned gains, (ii) the discrepancy between the hit probability p_{hit} measured by *ModelGraft* vs the one gathered through the event-driven approach remains always under 2%. It is worth remarking that, despite Eq. 4 proves the correctness of the TTL approximation under specific conditions (as also shown in [12], and remarked in [16, 29]), providing analytical lower/upper bounds of the estimation accuracy of *ModelGraft* for complex cache networks is not trivial; as a consequence, we resort in assessing *ModelGraft* accuracy by comparing its results with the correspondent LRU non-downscaled scenario simulated through event-driven simulation. Finally, it is interesting to point out the absence of either an (iii) accuracy vs. speed trade-off, as one would typically expect [34], or of a (iv) memory vs. CPU trade-off [21], i.e., cases where an algorithm either trades increased space (e.g., cached results) for decreased execution time (i.e., avoid computation), or viceversa. *ModelGraft* thus stands in a rare win-win operational point where both CPU and memory usage are significantly relieved, at a price of a very small accuracy loss.

7.2. Gain projection: Internet-scale Scenario

We finally employ the validated *ModelGraft* engine to venture scenarios that are prohibitively complex for classic event-driven simulation, due to both CPU and memory limitations. Indeed, our aim is to investigate Internet-scale scenarios, whose content catalogs are estimated to be in the order of $\mathcal{O}(10^{11})$ – $\mathcal{O}(10^{12})$ [33], i.e., *two orders of magnitude larger than those considered in the previous section.*

We consider two representative scenarios. The first one, depicted in Fig. 4(a), models an access network represented by a 4-level binary tree with a single repository connected to the root, serving a catalog with cardinality $M = 10^{10}$. Cache size is $C = 10^6$, which limits the maximum downscaling to $\Delta = 10^5$ (see [45] for an exhaustive sensitivity analysis on Δ). The second scenario, depicted in Fig. 4(b), models, instead, a more complex CDN network, where three repositories, serving a catalog with cardinality $M = 10^{11}$, are connected to backbone nodes interconnected as the classic Abilene network, and where an access tree is further attached to each backbone node, thus resulting in a total of $N = 67$ nodes. In this scenario, we let the cache size be $C = 10^7$, which allows to increase the downscaling factor to $\Delta = 10^6$. In both cases, considering the lack of ground truth due to prohibitive simulation cost, we provide rough T_C guesses as *ModelGraft* inputs.

As before, we set $Y = 0.75$, $\varepsilon_{CV} = 5 \cdot 10^{-3}$, and $\varepsilon_C = 0.1$, and we run experiments on the same machine. Although we cannot instrument event-driven simulations at such large-scale, due to both physical memory limits (a hard constraint), as well as time budget (a soft constraint), we can *estimate* the expected memory occupancy and CPU times by means of simple predictive models: despite pertaining only to the specific implementation of ccnSim (and so being of limited interest), such estimates are nevertheless useful to

Table 3: Internet-scale scenarios: *ModelGraft* results and projected gains vs event-driven simulation.

Topology	Parameters	Technique	p_{hit}	CPU time (#Cycles)	Gain	Mem	Gain
Access-like (N = 15)	M = 10^{10} R = 10^{10} C = 10^6 Δ = 10^5 Y = 0.75	Simulation (estimate)	n.a.	4.5 days	**270x**	70 GB	**~1500x**
		ModelGraft	**31.4%**	**24 min (1 cycle)**		**45 MB**	
CDN-like (N = 67)	M = 10^{11} R = 10^{11} C = 10^7 Δ = 10^6 Y = 0.75	Simulation (estimate)	n.a.	50 days	**96x**	520 GB	**~16700x**
		ModelGraft	**34.0%**	**12.5 h (3 cycles)**		**31 MB**	

project *ModelGraft* gains. While the CPU *time* is proportional[3] to the the total number of requests, the *memory occupancy* (Mem_{sim}) of the optimized event-driven module of ccnSim is well fitted by the following formula:

$$Mem_{sim} = \mu_{cache} NC + \mu_{catalog} M + \mu_{fix}, \tag{14}$$

where N is the number of nodes, C the cache size, M the catalog cardinality, while the three constants μ_{cache}, $\mu_{catalog}$, and μ_{fix}, represent the bytewise memory occupancy of individual entries of the cache, of individual entries of the catalog, and the fixed footprint of the simulator environment, respectively. By fitting the model over more than 50 scenarios (i.e., varying N, M, C and several other scenario parameters such routing, caching policies, etc.) we gathered $\mu_{cache} = 165\,\text{Bytes}$, $\mu_{catalog} = 4\,\text{Bytes}$, and $\mu_{fix} = 20 \cdot 10^6\,\text{Bytes}$, with asymptotic standard errors of 3.28%, 0.02%, and 0.03%. Notice that this formally confirms the catalog size (orders of magnitude larger than the cache size) to constitute the primary reason to prevent event-driven simulations at scales such as the ones considered in this paper.

Projected *ModelGraft* gains are reported in Tab. 3. Consider the Access-like scenario first: even though the rejection inversion sampling technique optimizes the memory allocation of both simulation and *ModelGraft*, however an index binding all the seed copies with their respective repositories is still needed: this scales as M in case of event-driven simulation, and with $M' = M/\Delta \ll M$ in *ModelGraft*, which explains why the projected memory gains in Tab. 3 (obtained for $\Delta = 10^6$) are significantly higher (i.e., more than three orders of magnitude) than the ones shown in Tab. 2 (obtained for $\Delta = 10^5$). Regarding CPU time, instead, relative gains are similar (i.e., higher than two orders of magnitude) to Tab. 2, as our initial T_C guesses (inferred from simple interpolation of smaller scale scenarios) were accurate enough to let *ModelGraft* converge. Considering the CDN-like scenario next, we notice that memory gains increase by one further order of magnitude (since in this case $\Delta = 10^7$), whereas CPU gains are slightly reduced (to a still large 96×) in reason of the higher number of MC-TTL cycles[4].

[3] Notice that this model is *conservative*, as it neglects superlinear effects such as, e.g., the cost of lookups in larger data structures: as such *ModelGraft* gains may be *underestimated.*

[4] Specifically, given that our initial T_C guesses were not accurate enough, *ModelGraft* performed three

Table 4: Sensitivity analysis parameters. Full details in [45]

Variable parameters

Parameter	*Default*	*Range*
T_C	T_C^{sim} (Tab. 5)	$[1/100, 100]T_C^{sim}$
Δ	10^5	$\{10^1, 10^2, 10^3, 10^4, 10^5, 10^6\}$
Y	0.75	$\{0.5, 0.75, 0.9, 0.95, 1\}$
α	1	$\{0.8, 1, 1.2\}$
ε_{CV}	0.005	$\{0.005, 0.01, 0.05, 0.1\}$
ε_C	0.1	$\{0.05, 0.1, 0.5\}$

Fixed parameters

Parameter	*Value*
M	10^9
C	10^6
R	10^9
Caching	$\{LCE, LCP(1/10)\}$
Topology	{4-level binary tree, NDN Testbed}

Table 5: T_C^{sim} values for validation scenario (4-level binary tree, $M = R = 10^9, C = 10^6, Y = 0.75$)

Level		**LCE**	**LCP(1/10)**	**2-LRU (Name/Main)**
0 (Root)	**T_C values** [s]	$16.7 \cdot 10^3$	$16.7 \cdot 10^4$	$20.0 \cdot 10^3$ / $76.4 \cdot 10^4$
1		$32.5 \cdot 10^3$	$31.4 \cdot 10^4$	$38.1 \cdot 10^3$ / $12.5 \cdot 10^5$
2		$63.0 \cdot 10^3$	$56.9 \cdot 10^4$	$71.9 \cdot 10^3$ / $20.4 \cdot 10^5$
3 (Leaves)		$11.1 \cdot 10^4$	$88.3 \cdot 10^4$	$11.1 \cdot 10^4$ / $22.6 \cdot 10^5$
p_{hit}		33.2%	35.4%	37.0%

8. Sensitivity analysis

We test the correctness of *ModelGraft* by varying all the parameters summarized in Tab. 4, in order to assess its gains under an extensive set of scenarios. To gather conservative estimate we consider the very large-scale scenario of Sec. 7.1 as a baseline. Furthermore, we make available all the scripts used to perform the sensitivity analysis at [2]. Finally, we point out that a complete account of the sensitivity analysis, carried over more than 340 experiments for cumulated 31 days of CPU time, is available in an extended technical report [45].

8.1. Input (T_C) sensitivity

As described in Sec. 6.1, *ModelGraft* is self-stabilizing: i.e., after reaching steady-state, some system properties (namely, the measured cache size $\tilde{C}$) are compared with the expected ones (namely, the downscaled target cache size C/Δ). In case of disagreement among these quantities, actions are taken by correcting a fundamental system property (namely, the characteristic time $T_{C_i}^{(z+1)}$ of each node, Sec. 6) before entering a new MonteCarlo simulation cycle.

As previously seen in Sec. 7.2, correctness of $T_{C_i}^{(0)}$ values provided as input may impact the number of MC-TTL cycles needed to reach a consistent state, thus affecting the overall CPU time. We can expect that in the typical *ModelGraft* use cases, the exact T_C values are not known a priori (as simulation is precluded in those scenarios): it is thus of primary importance to assess its performance in the presence of rough T_C

cycles to converge, lengthening the duration of the Monte-Carlo simulation and reducing the gains with respect to simulation.

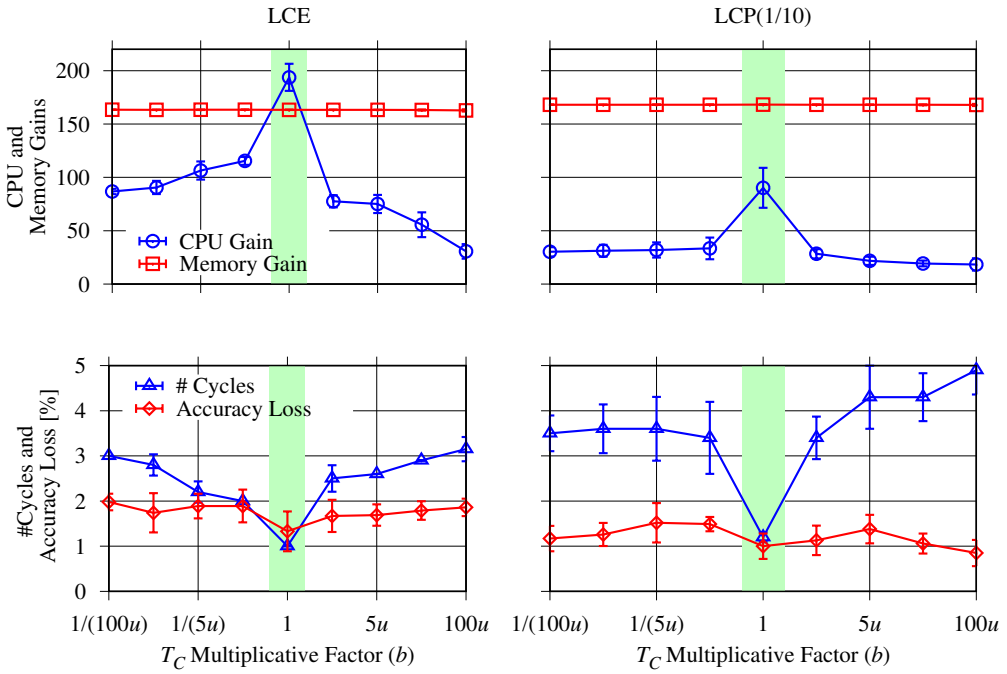
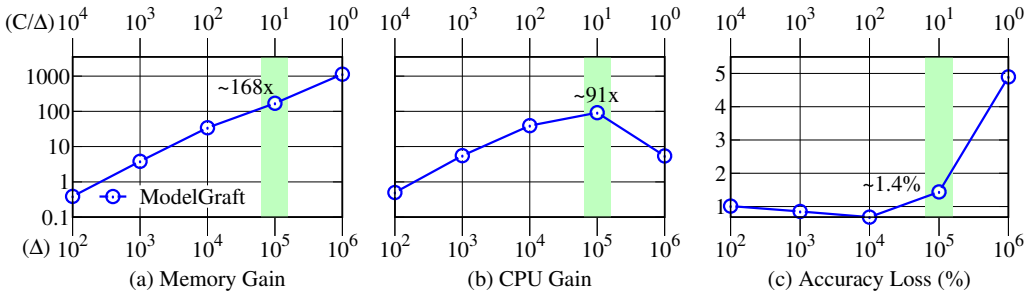


Figure 5: T_C sensitivity: 4-level binary tree, $M = 10^9, R = 10^9, C = 10^6, \Delta = 10^5$, for LCE (a,c) and LCP(1/10) (b,d) cache decision policies.

guesses. For the sake of clarity, we report in Tab. 5 T_C values gathered from event-driven simulations of the same scenario, for different cache decision policies. It can be clearly noticed that T_C values vary more according to the *cache decision policy* (up to one order of magnitude) than to the *topological position* inside the network; in particular, the more conservative the cache decision policy (e.g., LCP or 2-LRU), the bigger the T_C values. Intuitively, this reflects the fact that if nodes regulate the storage of new contents (as in LCP or 2-LRU), those already admitted in the cache will stay for longer periods (i.e., bigger T_C values). Also intuitively, these contents are more popular (i.e., note that for stationary content popularities, LCP(ε) converges to LFU for vanishing ε [29]), which, in turn, increases the overall hit probability (as the last row of Tab. 5 shows).

With this baseline in mind, our approach in testing the resilience of *ModelGraft* against input T_C variability consists in purposely introducing estimation errors, for each node, in a controlled fashion. In particular, we set $T_C^0(i) = b(i)T_C^{sim}(i)$, where $T_C^0(i)$ is the initial characteristic time for node i, $T_C^{sim}(i)$ is the accurate value (gathered via simulation and reported in Tab. 5), and $b(i) \in [1/(Bu), Bu]$ is a multiplicative factor obtained by multiplying a bias value $B \in [1, 100]$ (equal for all the nodes) by a uniform random variable $u \in (0, 1]$ (i.i.d. for all nodes). This means that we allow both *overestimating* (when $b(i) > 1$) and *underestimating* ($b(i) < 1$) the actual $T_C^{sim}(i)$ value. Notice that, in case of maximum bias (i.e., $B_{max} = 100$), $T_C^0(i)$ will differ from $T_C^{sim}(i)$ by *up to two orders of magnitude* (i.e., larger than the maximum difference previously observed),



(a) Memory Gain (b) CPU Gain (c) Accuracy Loss (%)

Δ	Simulation	*ModelGraft*
10^2	6404	16549
10^3	6404	1674
10^4	6404	187
10^5	6404	38
10^6	6404	6

(d) Memory[MB]

Δ	Simulation	*ModelGraft*
10^2	26280	52810
10^3	26280	4741
10^4	26280	666
10^5	26280	287
10^6	26280	4841

(e) CPU time [s]

Δ	Simulation	*ModelGraft*
10^2	35.40	34.39
10^3	35.40	34.55
10^4	35.40	34.72
10^5	35.40	33.96
10^6	35.40	30.50

(f) p_{hit}[%]

Figure 6: Downscaling factor Δ sensitivity: 4-level binary tree, with $M = R = 10^9, C = 10^6, Y = 0.75$, LCP(1/10) as cache decision policy: (a-c) Relative gains, (d-f) Tabulated absolute values. Default value highlighted in green.

varying furthermore for each node in the network due to the uniform random variable u.

Fig. 5 reports the collected results related to four KPIs (namely, CPU and memory gains, accuracy loss, and number of MC-TTL *cycles*), under the LCE and LCP(1/10) cache decision policies, as a function of the multiplicative factor b. Several remarks follow from Fig. 5. To begin with, (i) the presence of a feedback control loop efficiently drives the system to a consistent state, thereby nullifying the effects of the initial T_C variability: notice, indeed, that the accuracy loss on p_{hit} remains always under 2%, regardless of the magnitude of the bias $b(i)$. Additionally, (ii) MC-TTL cycles always remain bounded to a small integer, regardless of the cache decision policy, and even for initial T_C estimates that are off by more orders of magnitude, which confirms the soundness of the choice of a proportional controller in Sec. 6. It also follows that (iii) while CPU gains are maximum in the absence of T_C bias since no iterations are required (the central highlighted region, corresponding to about 200x and 100x for LCE and LCP, respectively), they still remain significant even for the largest T_C biases (always higher than 50x, for both LCE and LCP). Finally, (iv) memory gain is, as expected, independent of the initial T_C bias.

Remark 1. *We observe that* ModelGraft *is* self-stabilizing*, meaning that it is able to gather the metrics of interest not only* accurately*, but also* timely *(i.e., it rapidly converges), even when very harsh guesses of T_C values (e.g., T_C estimates that are wrong by* orders of magnitude*) are provided as input. Despite the largest memory and CPU time reduction are attained when exact T_C values are provided as input (as expected),* ModelGraft *converges within few cycles even with very inaccurate T_C estimates, still providing gains of* orders of magnitude *with respect to event-driven simulation. This makes the methodology very robust for practical purposes.*

8.2. Downscaling factor (Δ) sensitivity

Another key parameter having a paramount impact on *ModelGraft* performance, in terms of both CPU time and memory occupancy, is the *downscaling factor* Δ. We assess its impact using LCP(1/10) as cache decision policy, and providing *ModelGraft* with exact T_C estimates as input (cfr. Tab. 5). We vary the downscaling factor Δ in the interval $[10^2, 10^6]$, so that the target cache $C' = C/\Delta$ decreases from 10^4 to 10^0.

Fig. 6-(a-c) depict the three KPIs, i.e., memory gain $Mem^{Simulator}/Mem^{ModelGraft}$, CPU time gain $CPU^{Simulator}/CPU^{ModelGraft}$, and accuracy loss $|p_{hit}^{Simulation} - p_{hit}^{ModelGraft}|$, while the corresponding absolute values are tabulated in Fig. 6-(d-f). Several remarks are in order. First, note that (i) for small downscaling factors $\Delta \approx 10^2$, the scalability is actually *compromised*: notice, indeed, that both memory occupancy and CPU time *exceed* that of classic event-driven simulation. This is mainly due to the absence of size limits in TTL caches, which, despite reaching their target size $C' = C/\Delta$ only at steady-state, present peaks with $C' > C$ during the transient state, thus resulting in a bigger memory occupancy with respect to LRU caches. In addition, as lookup in TTL caches cannot leverage an ordered structure, as it would in LRU ones, it follows that larger caches also translate into a higher CPU time. Without *ModelGraft*, then, TTL caches would not allow, per se, the study of large scale systems.

Second, notice that (ii) for increasing Δ, memory and CPU gains increase, while accuracy losses remain bounded: *gains are maximized for* $\Delta \approx C/10$, which in the current scenario corresponds to $\Delta = 10^5$. Notice that while this section reports a specific case, we have experimentally observed the same trend for a wider set of scenarios, so that we are confident in using this rule of thumb as a robust and reliable guideline for setting Δ. This also means that the larger the cache size, the larger the allowed downscaling factor Δ, and the larger the gain – which we have early shown in Sec. 7.2.

Third, observe that (iii) for $\Delta \approx C$, *ModelGraft* becomes *unstable*, which happens because the target cache size $C' \approx 1$ is too small to be reliably measurable. The effect is further amplified by the series of controller actions in (13), with an error that possibly amplifies:

$$T_{Ci}{}^{(z+1)} = T_{Ci}{}^{(0)} \prod_{j=0}^{z} \frac{C'}{\widetilde{C}_i^{(j)}}. \tag{15}$$

As the very small target cache $C' = 1$ hampers the measurement step of the consistency check, this results in additional MC-TTL cycles: e.g., in the specific case of Figs. 6-(b-c), *ModelGraft* simulations reach the maximum number of cycles (set to 20). Such unstable states are, however, easy to detect (memory occupancy still decreases, while execution time increases) and avoid (as per the previous Δ dimensioning rule).

Remark 2. *The downscaling factor Δ has to be chosen in accordance with the cache size C. The system is stable for a large plateau of Δ values up to $\Delta^\star = C/10$, where largest gains are expected: for $\Delta < \Delta^\star$ the system is still stable but gain reduces, whereas for $\Delta > \Delta^\star$* ModelGraft *becomes unstable. For $\Delta \leq 10^2$, the memory requirement of TTL caches exceeds that of classic LRU event-driven simulation, which makes our downscaling technique highly important from practical purposes.*

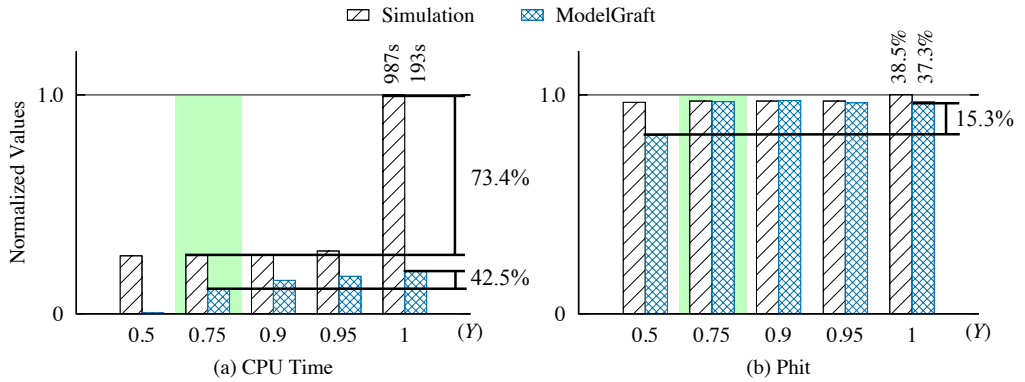


Figure 7: Consistency parameter Y sensitivity: very large-scale scenario for NDN Testbed topology with, $M = R = 10^9, C = 10^6, \Delta = 10^5$, LCE. Normalized (a) CPU time and (b) p_{hit} when the system converges to steady-state. Default value highlighted in green.

8.3. Consistency parameter (Y) sensitivity

Another parameter that impacts *ModelGraft* CPU time is represented by Y. As seen in Sec. 5.2, transient periods have, in *ModelGraft*, a variable duration depending on the state of individual nodes: Y allows to tune the number of nodes that are considered for the convergence computation, with the purpose of avoiding waiting for few slower nodes (e.g., the ones with low traffic and small measured caches, etc.), which, in the end, have a marginal impact on the overall system performance. We expect the system to be quite robust to Y, as this parameter only affects the time at which the system enters steady-state, after which a number R' of events is executed to gather the metrics of interest over all network nodes.

While for hierarchical topologies it is immediate to identify which nodes are slower to converge (e.g., in the 4-level binary tree topology depicted in Fig. 4(a), the latest node to converge is represented by the root, that aggregates the miss stream of request for unpopular content that are not filtered by cache hit in downstream caches), this is no longer the case for more complex topologies or routing strategies. We investigate this aspect by considering the very large-scale scenario ($M = 10^9, R = 10^9, C = 10^6, \Delta = 10^5$, LCE), but using the NDN Testbed topology [3] with $N = 26$ nodes. We purposely vary the Y parameter in the interval $[0.5, 1]$, for both *ModelGraft* and classic event-driven simulation, requiring convergence of the first YN nodes in each *ModelGraft* cycle.

We report the convergence time in Fig.7-(a) , i.e., the elapsed time at which the simulation is considered at steady-state, and the measured hit rate p_{hit} in Fig. 7-(b) at that time instant. Both figures report normalized values with respect to the maximum CPU value, and to the ground truth value of p_{hit} gathered from event-driven simulation for $Y = 1$, respectively. The figure clearly shows that Y has a beneficial impact on execution time without hurting accuracy for a large plateau of values. Specifically (i) the largest CPU time gain is obtained when Y passes from 1 to 0.95 (i.e., eliminating the single slowest node to converge reduces most of the unnecessary wait), that has only a minimal and conservative impact on the accuracy (as the hit probability only vary slightly decreases). Moreover, (ii) the CPU gain increases further until $Y = 0.75$ with no additional impact on the accuracy. Finally, (iii) accuracy degrades only for extreme

Table 6: Content popularity α sensitivity: very-large scenario, 4-level tree, $M = R = 10^9, C = 10^6, \Delta = 10^5, Y = 0.75$, LCE.

α	Technique	p_{hit}	CPU time	Gain	Mem [MB]	Gain
0.8	Simulation	7.96%	22.8 h	**461x**	6356	**168x**
	ModelGraft	7.85%	179 s		38	
1	Simulation	33.2%	11.4 h	**194x**	6371	**168x**
	ModelGraft	31.4%	211 s		38	
1.2	Simulation	52.9%	4.2 h	**52x**	6393	**168x**
	ModelGraft	52.6%	286 s		38	

cases where only $Y = 0.5$ half of the nodes have reached convergence.

Remark 3. *The spatial sampling parameter Y impacts convergence speed and accuracy. It allows to accelerate steady-state achievement for $Y<1$: already excluding only the slowest node from the convergence test, i.e., $Y \approx \frac{N-1}{N}$, provides significant computational gains. Further computational gains are possible, although they may come at the price of accuracy loss when $Y < 0.75$.*

8.4. Content popularity (α) sensitivity

Finally, we verify *ModelGraft* consistency in different scenarios, by varying the exponent $\alpha \in \{0.8, 1, 1.2\}$ of the Zipf distribution . Tab. 6 reports numerical values and relative gains of *ModelGraft* with respect to event-driven simulation. As expected, the popularity skewness does not heavily influence *ModelGraft* performance: very limited accuracy losses of 0.1% and 0.3% appear for $\alpha = 0.8$ and $\alpha = 1.2$, respectively, with a memory reduction of 168× in both scenarios. Interestingly, the CPU gain for the case $\alpha = 0.8$ is more than twice the one obtained with $\alpha = 1$ (i.e., 461× instead of 194×): *ModelGraft* simulations last for 3 minutes, instead of almost 1 day for the event-driven engine. For the case $\alpha = 1.2$, instead, the computational gain is smaller but still significant (52×). This can be explained with the fact that in this scenario, a hit ratio higher than 50% translates in very limited propagation of content requests (i.e., generated events), since most requests are satisfied at leaf caches; as such, the CPU time of the classic event-driven approach is already small per se, and the gain shrinks.

Remark 4. *The popularity distribution of the content catalog does not affect the accuracy of* ModelGraft*. In particular, CPU gains are larger for lower skews (close to 500x for $\alpha = 0.8$), but they remain consistent also for higher skews (over 50x for $\alpha = 1.2$), confirming* ModelGraft *scalability.*

8.5. Further parameters

Other parameters, such as convergence ε_{CV} and consistency ε_C thresholds, scenario topology and scale, etc. might influence *ModelGraft* execution time and accuracy. Due to lack of space, a brief discussion is reported herein, while the interested reader is referred to the technical report [45] for further details. As shown in Tab. 4, we tested $\varepsilon_{CV} \in \{0.005, 0.01, 0.05, 0.1\}$: as expected, the bigger ε_{CV}, the faster *ModelGraft* converges,

without affecting the accuracy at the same time. Notice that our selection $\varepsilon_{CV} = 0.005$ in the previous sections implies that we compare *ModelGraft* against event-driven simulation *gathering a conservative gain estimation.* We test the effect of the consistency threshold for $\varepsilon_C \in \{0.05, 0.1, 0.5\}$: we gather that small values (e.g., $\varepsilon_C = 0.05$) force *ModelGraft* to *unnecessarily* execute multiple cycles, i.e., even when *exact* T_C value are provided as input – which we avoid by letting $\varepsilon_C = 0.1$ as default.

Remark 5. *Estimated gains of our novel methodology over classic event-driven simulation provided in this paper are* rather conservative. *Notice, indeed, that since the very large-scale scenario limits, de facto, the downscaling factor to* $\Delta \leq 10^5$*, the* ModelGraft *CPU and memory gains reported in this section are conservative by design, considering that the methodology is designed to cope with extremely large scenarios, and that gains grow with* Δ *(Remark 2). Similarly, default settings of* ε_{CV} *and* ε_C *reported in this paper provide a conservative estimate of* ModelGraft *gains.*

9. Related Work

We now put our proposal in perspective w.r.t three classes of work that relate to ours.

Hybrid techniques for the study of general networks. The concept of inferring key aspects of large systems from the study of equivalent and scaled-down versions has been adopted in several domains, from cosmology and biology, to the closer IP networks [37, 28], wireless sensor networks [27], and control theory [8].

What presented in this paper shares the spirit of [37], where the idea of feeding a suitable scaled version of the system with a sample of the input traffic is presented to reduce the computational requirements needed for large IP networks analysis. In particular, the scaling rule is differentiated according to the type of TCP/UDP flows traversing the network: for IP networks with short and long flows they demonstrate, both analytically and with simulation, that their scaling technique leaves certain metrics, such as the distribution of the number of active flows and of their normalized transfer time, virtually unchanged in the scaled system. For networks with long-lived flows controlled by queue management schemes, a different scaling approach, instead, leaves queuing delay and drop probability unchanged. In this latter case, the proposed approach drastically reduces the CPU time of *ns* simulations.

TCP networks are also considered in [28], where the authors propose a scalable model which is easily comparable with discrete event simulators due to its time-stepped nature. In particular, they refine a known analytical model [36] based on ordinary differential equations, and they solve it numerically using the Runge-Kutta method. Results show that their approach yields accurate results with respect to those of the original networks, and, at the same time, it is able to speedup the completion time of orders of magnitude with respect to packet level and discrete events simulators like *ns*.

This work is the first to apply these concepts to the study of cache networks. Clearly, caching dynamics are intrinsically different from system-level aspects of wireless sensor networks [27], or the steady state throughput of TCP/IP networks [37, 28]. Our methodology is novel with respect to the body of hybrid approaches [8, 36, 27, 28, 37] in that, unlike our approach, they do not allow to study network of caches, that have become a crucial piece of the current (CDN) or future (ICN) Internet.

General tools for the study of cache networks. Additionally, an aspect that makes our work of high practical relevance is that *ModelGraft* is fully integrated in ccnSim [2], an open-source tool that was already among the most scalable ones [43], and even improving its by orders of magnitude. We notice that, despite general techniques to scale simulation tools do exist, they however focus on orthogonal problems with respect to ours. For instance, Message Passing Interface (MPI) [1], SimGrid [10], and Akaroa [14] allow to parallelize the simulation execution.

In particular, we already used MPI to parallelize ccnSim [40] by simply slicing the network: however, this is not particularly attractive in cache networks, due to the correlation of states among neighboring caches, which makes the overhead of MPI offsetting any benefit [40]. Also, in previous sections we have shown memory and computational bottlenecks to be tied to the vastness of objects in the catalog, which is comparatively larger than the state of network caches – reinforcing the need of a technique such as the one we propose and analyze in this paper.

Analytical tools for the study of cache networks. Finally, while our technique leverages existing building blocks, namely Che's approximation [12] and TTL caches [23, 16, 30, 32], it does however bring a number of original contributions. First, while we leverage Che's intuition to decouple objects in the catalog, however the technique to opportunely subsample the catalog is an original contribution of this work – that enables sizable gains of over 100x CPU time reduction, and over 1000x memory footprint reduction in Internet-like scenarios. We indeed remark that TTL caches with small downsizing are computationally more costly than classic LRU ones in event-driven simulation. This follows from the fact that TTL caches are not bounded, so that when $\Delta \approx 1$, the CPU time of TTL-based simulation actually exceeds that of classic event-driven simulation [45]. It follows that the use of TTL caches[23, 16, 30, 32] would be impractical in Internet-scale scenarios without our proposed downscaling technique. This also explains why, though intellectually interesting, TTL caches have been so far used only for very simple networks (e.g., single cache, trees) and catalogs (few hundreds objects), thus reinforcing the relevance of our proposal.

Additionally, while we leverage TTL caches, another original contribution is to formalize their equivalence with LRU systems, and to further design a closed-loop system able to accurately converge to performance estimates with the aforementioned gains. Notice, *en passant*, that our technique copes with other replacement policies than LRU, such as, e.g., random replacement (RND) or FIFO, as extensions of Che's approximation [17] and networks of LRU caches, are largely more popular in practice. Similarly, the support for arbitrary routing and forwarding techniques on complex topologies makes *ModelGraft* very relevant from practical perspectives.

10. Conclusion

This work proposes *ModelGraft*, an innovative hybrid methodology addressing the issue of performance evaluation of large-scale distributed cache networks. The methodology grafts elements of stochastic analysis to MonteCarlo simulation approaches, retaining benefits of both classes. Indeed, *ModelGraft* inherits simulation *flexibility*, in that it can address complex scenarios (e.g., topology, cache replacement, decision policy, etc.).

Additionally, it is implemented as a simulation engine to retain simulation *simplicity*: given its self-stabilization capability, *ModelGraft* execution is decoupled from the availability of accurate input T_C values, which is completely transparent to the users. Results presented in this paper finally confirm both the *accuracy* and the *high scalability* of the *ModelGraft* approach: CPU time and memory usage are reduced by (at least) two orders of magnitude with respect to the classical event-driven approach, while accuracy remain within a 2% band.

Whilst the methodology is general, we offer to the scientific community an open-source implementation in ccnSim-v0.4, which is readily available in [2]. As *ModelGraft* requires little tuning, users can leverage, in a seamless way, either the classic event-driven engine or the new *ModelGraft*, which empowers their analysis of Internet-scale scenarios.

Acknowledgments

This work benefited from support of NewNet@Paris, Cisco's Chair "NETWORKS FOR THE FUTURE" at Telecom ParisTech (`http://newnet.telecom-paristech.fr`). Any opinion, findings or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of partners of the Chair.

References

[1] `http://www.mpi-forum.org/`.
[2] ccnSim Simulator. `http://perso.telecom-paristech.fr/~drossi/ccnSim`.
[3] NDN Testbed web page. `http://named-data.net/ndn-testbed/`.
[4] Zipf distributed random number generator . `https://github.com/apache/commons-math/blob/master/src/main/java/org/apache/commons/math4/distribution/ZipfDistribution.java`.
[5] A. Araldo, D. Rossi, et al. Cost-aware caching: Caching more (costly items) for less (isps operational expenditures). *IEEE Transactions on Parallel and Distributed Systems,*, 2015.
[6] S. Arianfar and P. Nikander. Packet-level Caching for Information-centric Networking. In *ACM SIGCOMM, ReArch Workshop*. 2010.
[7] F. Baccelli, S. Machiraju, et al. On optimal probing for delay and loss measurement. In *Proc. of ACM IMC*. 2007.
[8] M. Branicky, V. Borkar, et al. A unified framework for hybrid control: model and optimal control theory. *IEEE Transactions on Automatic Control*, 43(1):31, 1998.
[9] G. Carofiglio, L. Mekinda, et al. Analysis of latency-aware caching strategies in information-centric networking. In *ACM CoNEXT Workshop on Content Caching and Delivery in Wireless Networks (CCDWN '16)*. 2016.
[10] H. Casanova, A. Giersch, et al. Versatile, scalable, and accurate simulation of distributed applications and platforms. *Journal of Parallel and Distributed Computing*, 74(10):2899, 2014.
[11] M. Cha, H. Kwak, et al. I tube, YouTube, everybody tubes: analyzing the World's largest user generated content video system. In *Proc. of ACM IMC*. 2007.
[12] H. Che, Y. Tung, et al. Hierarchical web caching systems: Modeling, design and experimental results. *IEEE JSAC*, 20(7):1305, 2002.
[13] A. Dan and D. Towsley. An Approximate Analysis of the LRU and FIFO Buffer Replacement Schemes. *ACM SIGMETRICS Perf. Eval. Rev.*, 18(1):143, 1990.
[14] G. Ewing, K. Pawlikowski, et al. Akaroa-2: Exploiting network computing by distributing stochastic simulation. *SCSI Press*.
[15] R. Fagin. Asymptotic miss ratios over independent references. *Journal of Computer and System Sciences*, 14(2):222, 1977.
[16] N. Fofack, P. Nain, et al. Performance evaluation of hierarchical TTL-based cache networks. *Elsevier Computer Networks*, 65, 2014.
[17] C. Fricker, P. Robert, et al. A versatile and accurate approximation for LRU cache performance. In *Proc. of ITC24*. 2012.

[18] M. Garetto, E. Leonardi, et al. Efficient analysis of caching strategies under dynamic content popularity. In *Proc. of IEEE INFOCOM*. 2015.
[19] N. Gast and B. V. Houdt. Transient and steady-state regime of a family of list-based cache replacement algorithms. In *Proc. of ACM SIGMETRICS Conference*, pages 123–136. 2015.
[20] M. Hefeeda and O. Saleh. Traffic modeling and proportional partial caching for peer-to-peer systems. *IEEE/ACM Transactions on Networking*, 16(6):1447, 2008.
[21] M. E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401, 1980.
[22] W. Hörmann and G. Derflinger. Rejection-inversion to generate variates from monotone discrete distributions. *ACM Trans. Model. Comput. Simul.*, 6(3):169, 1996.
[23] J. Jaeyeon, A. W. Berger, et al. Modeling TTL-based Internet caches. In *Proc. of IEEE INFOCOM*. 2003.
[24] T. Johnson, D. Shasha, et al. 2q: A low overhead high performance buffer management replacement algorithm. In *Proc. of VLDB*. 1994.
[25] N. Laoutaris, H. Che, et al. The LCD interconnection of LRU caches and its analysis. *ACM SIGMETRICS Perf. Eval. Rev.,*, 63(7), 2006.
[26] E. Leonardi and G. Torrisi. Least Recently Used caches under the Shot Noise Model. In *Proc. of IEEE INFOCOM*. 2015.
[27] P. Levis, N. Lee, et al. TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. In *Proc. of ACM SenSys*. 2003.
[28] Y. Liu, F. Presti, et al. Scalable Fluid Models and Simulations for Large-scale IP Networks. *ACM Trans. Model. Comput. Simul.*, 14(3):305, 2004.
[29] V. Martina, M. Garetto, et al. A unified approach to the performance analysis of caching systems. In *Proc. of IEEE INFOCOM*. 2014.
[30] D. Berger et al. Exact Analysis of TTL Cache Networks: The Case of Caching Policies Driven by Stopping Times. In *Proc. of ACM SIGMETRICS*, pages 595–596. 2014.
[31] S. Fayazbakhsh et al. Less Pain, Most of the Gain: Incrementally Deployable ICN. *ACM SIGCOMM Comput. Commun. Rev.*, 43(4), 2013.
[32] N. Fofack et al. On the performance of general cache networks. In *Proc. of VALUETOOLS Conference*, pages 106–113. 2014.
[33] K. Pentikousis et al. Information-centric networking: Evaluation methodology. Internet Draft, https://datatracker.ietf.org/doc/draft-irtf-icnrg-evaluation-methodology/, 2016.
[34] M. Rosenblum et al. Complete Computer System Simulation: The SimOS Approach. *IEEE Parallel Distrib. Technol.*, 3(4):34, 1995.
[35] G. Xylomenos et al. A survey of information-centric networking research. *IEEE Communication Surveys and Tutorials,*, 16(2):1024, 2014.
[36] V. Misra, W. Gong, et al. Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED. *ACM SIGCOMM Comput. Commun. Rev.*, 30(4):151, 2000.
[37] R. Pan, B. Prabhakar, et al. SHRiNK: A Method for Enabling Scaleable Performance Prediction and Efficient Network Simulation. *IEEE/ACM Trans. Netw.*, 13(5):975, 2005.
[38] E. Rosensweig, D. Menasche, et al. On the steady-state of cache networks. In *Proc. of IEEE INFOCOM*. 2013.
[39] E. J. Rosensweig, J. Kurose, et al. Approximate Models for General Cache Networks. *Proc. of IEEE INFOCOM*, 2010.
[40] G. Rossini and D. Rossi. ccnSim: a highly scalable CCN simulator. In *Proc. of IEEE ICC*. 2013.
[41] G. Rossini and D. Rossi. Evaluating CCN Multi-path Interest Forwarding Strategies. *Comput. Commun.*, 36(7):771, 2013.
[42] G. Rossini and D. Rossi. Coupling caching and forwarding: Benefits, analysis, and implementation. In *Proc. of ACM ICN*. 2014.
[43] M. Tortelli, D. Rossi, et al. ICN software tools: survey and cross-comparison. *Elsevier Simulation Modelling Practice and Theory*, 63:23 , 2016.
[44] M. Tortelli, D. Rossi, et al. ModelGraft: Accurate, Scalable, and Flexible Performance Evaluation of General Cache Networks. In *Proc. of ITC*. Würzburg, Germany, 2016.
[45] M. Tortelli, D. Rossi, et al. Modelgraft: Accurate, scalable, and flexible performance evaluation of general cache networks (extended version). Telecom ParisTech Tech. Rep., `http://www.enst.fr/~drossi/paper/ModelGraft.pdf`, 2016.