ModelGraft: Accurate, Scalable, and Flexible Performance Evaluation of General Cache Networks

ModelGraft: Accurate, Scalable, and Flexible Performance Evaluation of General Cache Networks

Michele Tortelli*, Dario Rossi*, Emilio Leonardi†
* Télécom ParisTech, Paris, France † Politecnico di Torino, Torino, Italy

***Abstract*—Large scale deployments of general cache networks, such as Content Delivery Networks or Information Centric Networking architectures, arise new challenges regarding their performance evaluation for network planning. On the one hand, analytical models can hardly represent in details all the interactions of complex replacement, replication, and routing policies on arbitrary topologies. On the other hand, the sheer size of networks and content catalogs makes event-driven simulation techniques inherently non-scalable.**

We propose a new technique for the performance evaluation of large-scale caching systems that intelligently integrates elements of stochastic analysis within a MonteCarlo simulative approach, that we colloquially refer to as *ModelGraft*. Our approach (i) leverages the intuition that complex scenarios can be mapped to a simpler equivalent scenario that builds upon Time-To-Live (TTL) caches; it (ii) significantly downscales the scenario to lower computation and memory complexity, while, at the same time, preserving its properties to limit accuracy loss; finally, it (iii) is simple to use and robust, as it autonomously converges to a consistent state through a feedback-loop control system, regardless of the initial state.

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%. In addition, we show that ModelGraft extends performance evaluation well beyond the boundaries of classic approaches, by enabling study of Internet-scale scenarios with content catalogs comprising *hundreds of billions* objects.

I. Introduction

Caching systems, from their simplest single-cache incarnation to more complex general networks, have attracted remarkable attention over the years. Different approaches have been proposed to study their performance, from exact analytical models [17] with a high computational cost, to refined approximations able to reduce the computational cost while preserving their accuracy within acceptable bounds [8, 15, 9, 18, 32, 7, 10, 31, 25, 22]. However, with Content Distribution Networks (CDNs) first, and then with the advent of a new networking paradigm, namely Information Centric Networking (ICN) [28], caches become the atomic part of globally deployed networks. Under CDN/ICN architectures, several factors like content replacement algorithms, cache decision policies, and forwarding strategies, interact with each other at a global scale: such intricate dependencies heavily influence the performance of the whole system, thus making it hard to rely only on pure analytical models to accurately predict network key performance indicators (KPIs).

It follows that, especially as a first step to assess the performance of new complex protocols, event-driven simulation techniques represent an appealing alternative: indeed, simulative techniques make it simpler to describe algorithmic interactions between all the different entities with the desired level of fidelity. At the same time, large-scale simulations require massive computational resources (both CPU and memory), to the point that the very same fidelity of simulation results may be compromised by the intrinsic scalability limit of the technique itself.

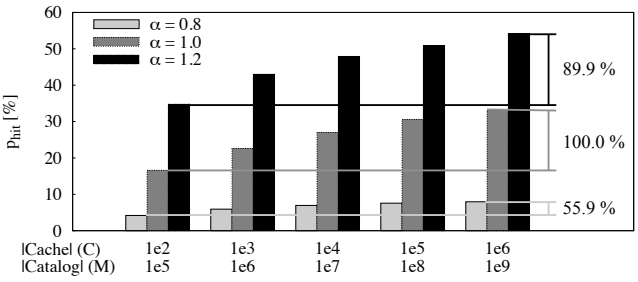


Fig. 1. Hit ratio variability - 4-level binary tree, variable α, fixed $C/M = 0.01\%$ ratio, with proportional variation of C and M.

To show why this may be the case, we provide in Fig. 1 an illustrative example. Whereas it is well known that Internet-scale catalogs approaches trillion objects [26], these scales are hardly achievable in simulation: as such, simulation-based studies typically resort to naïve downscaling of the scenario under investigation, to comply with memory constraints and CPU time budgets. Using the mean hit ratio as KPI, Fig. 1 contrasts the performance of a 4-level binary tree where the Zipf exponent is varied as $\alpha \in \{0.8, 1, 1.2\}$, and the ratio between catalog cardinality M and cache size C 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 does not preserve its original properties at all: the relative error between the smallest vs largest scenario is between 50% and 100%. This fact has profound implications: indeed, rather typically, crucial parameters of the scenario under investigation, such as the Zipf exponent α (or Mandelbrot Zipf plateau q), are measured over real Internet catalogs like YouTube [6] (or the BitTorrent ecosystem [12]) which are not only growing, but already had a catalog in the order of hundred millions objects about 10 years ago [6]. Clearly, Fig. 1 also implies that merely applying catalog parameters inferred from large-scale measurements to

small-scale simulations does not make any sense, as it induces excessive distortion of the KPIs to make results of practical relevance.

Inspired by *hybrid models* proposed over the years in different domains (see Sec. IX), we argue that *grafting* components of stochastic modeling into simulative techniques can increase the overall scalability by orders of magnitude, without compromising the accuracy of the resulting technique, at the same time. Specifically, our methodology exploits a synergy between stochastic analysis of Least Recently Used (LRU) caches [7, 22] and MonteCarlo approaches based on Time-To-Live (TTL) caches [15, 9, 23, 25]. In particular, our intuition consists in using the characteristic time T_C of Che's approximation [7] for LRU caches as the TTL parameter in MonteCarlo simulations: as a consequence, the complexity is significantly reduced by simulating TTL-based caches in place of their more complex LRU counterpart, and by subsampling the original catalog in a way that preserves its key statistical properties. Given that T_C in complex scenarios is not known a priory, our system uses a feedback loop to iteratively converge to the correct T_C value, even when the initial guess is wrong by two orders of magnitude.

We develop this intuition further to build a fully integrated system, making the following contributions:

- we propose a novel hybrid methodology for the performance evaluation of cache networks, that reduces CPU time and memory usage by over two orders of magnitude, while limiting accuracy loss to less than 2%;
- we implement the methodology as an alternative simulation engine, so that users can seamlessly switch between event-driven vs ModelGraft simulation, on the very same scenario;
- we make the technique (and scenarios) available as open source in the latest release of ccnSim [1].

In the remainder of this paper, we first provide background information about the building blocks leveraged by our methodology (Sec. II). We next provide a succinct overview of our proposal (Sec. III) followed by an in-depth description of each component (Sec. IV–VI). We next validate the technique (Sec. VII) and showcase its performance in very large scenarios (Sec. VIII), before covering related work (Sec. IX), and concluding the paper (Sec. X).

II. Modeling intuition

In this section we first introduce background material on Che's approximation [7] (Sec. II-A), and then formalize our intuition about the equivalence between LRU caches and (opportunely configured) TTL-based caches [15, 9, 23, 25] (Sec. II-B), which constitutes a fundamental building block of our methodology.

A. Background

Che's approximation [7], which emerged in the last few years as one of the most flexible models for cache networks, is essentially a mean-field approximation which greatly simplifies interactions between different contents inside a cache. In particular, for an LRU cache, Che's approximation consists in confusing the *cache eviction time* $T_C(m)$ for content m (i.e., the time since the last request after which object m will be evicted from the cache, unless the object is not requested again in the meantime), with a *constant characteristic time* T_C, which is a property of the whole cache but does not depend on the content itself. 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 an Independent Request Model (IRM), with content catalog of size M, and request process for content m to be Poisson of rate λ_m, the probability $p_{in}(m)$ for content m to be in a LRU cache at time t can be expressed as:

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

Since, by construction, cache size C must satisfy:

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

where $\mathbb{1}_{\{\mathrm{A}\}}$ is the indicator function for event A, then the characteristic time T_C can be computed by numerically inverting (2), which admits a single solution [7]. Finally, KPIs of the system, such as the *cache hit probability* p_{hit}, can be computed using the PASTA property as:

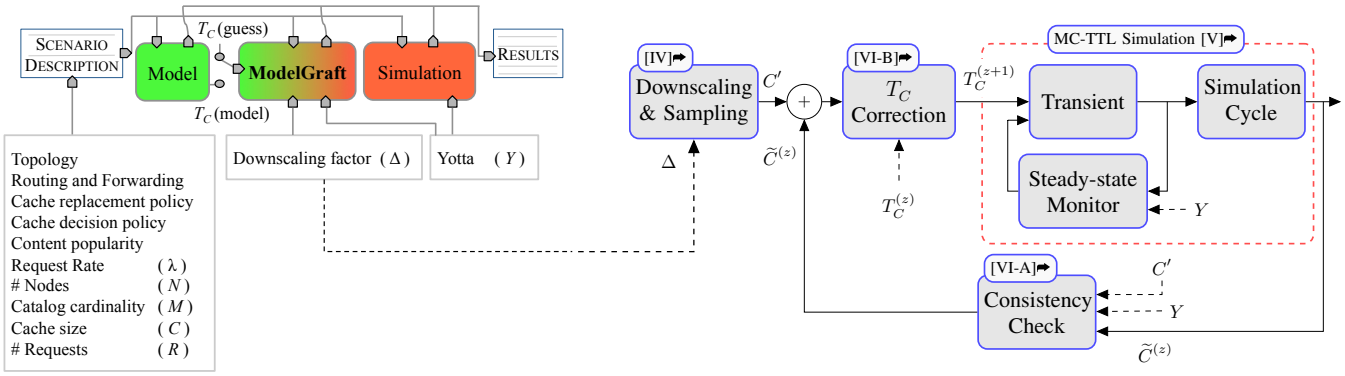
$$p_{hit} = \mathbb{E}_\Lambda[p_{in}(\lambda_m, T_C)] = \sum_m \lambda_m p_{in}(\lambda_m, T_C) / \sum_m \lambda_m \tag{3}$$

Although apparently simple, a theoretical explanation for the accuracy of Che's approximation under the IRM model has been provided only recently in [10], which shows that $T_C(m)$ converges to a constant independent of m as the cache grows large, and extended in [19] to renewal request models. As far as a single cache is concerned, Che's approximation was originally proposed for LRU caches [7], but it has been extended in more recent times to FIFO or Random replacement [10], or LRU caches with probabilistic insertion [22], possibly depending on complex cost functions [3].

As far as network of caches are concerned, however, further approximations are required as an alternative approach to the computationally and algorithmically challenging characterization of the miss stream at any node in the network [22, 25]. In arbitrary networks with shortest path [32] or more complex routing policies [33], it has been shown that inaccuracies can potentially cascade, with significant degradation of the accuracy with respect to simulation [34]. Finally, analytical approaches often assume stationary conditions, thus lacking in characterizing transient periods, although a model has been recently proposed only for a single cache [11]. All these reasons thus justify the quest for an hybrid approach, such as the one we propose in this work.

B. Intuition

Observe that, given the *characteristic time* T_C under Che's approximation, the dynamics of the different contents become completely decoupled. As a consequence, we can resort to simpler caching than LRU to simplify the analysis of complex and large cache networks. One such alternative is constituted by Time-to-Live (TTL) based caches [15, 9, 23, 25]: contents



(a) High level view of the integrated simulation workflow

(b) Details of the ModelGraft MonteCarlo TTL-Based workflow

Fig. 2. ModelGraft overview: (a) integration in ccnSim and (b) synoptic of the ModelGraft workflow.

are evicted from the cache after a pre-configured *eviction time* (also called TTL parameter) T_C' since the last request, unless a cache hit happens in the meantime (which resets the object timer).

Observation 1. *We argue that, the dynamics of a LRU cache with characteristic time T_C, fed by an IRM process associated to a catalog $\mathcal{M}$ with cardinality M, and request rates λ_m drawn from a distribution Λ, become indistinguishable from those of a TTL based cache with deterministic TTL parameter set equal to T_C operating on the same catalog*:

$$p_{hit}^{TTL}(T_C) = \mathbb{E}_\Lambda[1 - e^{-\lambda_m T_C}] = \mathbb{E}_\Lambda[p_{in}(\lambda_m, T_C)] = p_{hit}^{LRU}(T_C) \tag{4}$$

Specifically, (4) equals the average hit probability of the original LRU system to that of its TTL-based equivalent [9, 22]. Leveraging further on this intuition, we argue:

Observation 2. *Large-scale LRU networks can be analyzed through a downscaled system with $M' \ll M$, where each cache is replaced by its TTL equivalent, with TTL set to the characteristic time T_C of the original LRU cache. However, it is necessary to maintain the stochastic properties of the original catalog $\mathcal{M}$, while downscaling it. This, in turn, requires to average system performance over multiple MonteCarlo realizations, each lasting for a duration δT, where rates λ_m' for individual objects in each realization $\mathcal{M}'$ of the downsized catalogs are drawn from Λ. Thus, expanding (4):*

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

We expect, therefore, that decoupling the dynamics of different contents would allow to downscale the system, thus significantly reducing both memory and CPU complexity on the one hand, while still accurately representing complex interactions and correlations among different caches at the same time.

Observation 3. *In particular, an alternative approach is to let $\delta T \to 0$, and a convenient approximation is to vary λ_m' at each new request, still satisfying (4).*

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

III. ModelGraft overview

The hybrid methodology presented herein, colloquially referred to as ModelGraft, performs MonteCarlo simulations of an opportunely downscaled system, where LRU caches are replaced by their Che's approximated version, implemented in practice as TTL based caches. Before dwelving into the details of the approach, it is worth to both placing it into a broader context, as well as illustrating at high level each of its building blocks.

We implement ModelGraft as a simulation engine available in the latest version of ccnSim [1]. As illustrated in Fig.2(a), starting from a unique scenario description, users can analyze the performance of cache networks via either an *analytical model* [22] (left), a classic *event-driven* simulation engine (right), or via the *ModelGraft* engine (middle). ModelGraft depends on a single additional parameter, namely the *downscaling factor* Δ, which can be automatically set, or it is anyway very easy to tune (according to guidelines in Sec.VI-B).

As introduced in Sec. II-B, ModelGraft requires in input, T_C values for each cache in the network. 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)), which would, however, limit the appeal of the methodology. 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 the T_C value. In other words, ModelGraft is conceived to *auto-regulate*, so that by design it achieves accurate results even when the input T_C values, that the user does not even need to be aware of, largely differ from the correct ones.

Details of this iterative design are exposed in Fig.2(b), showing each of the blocks that are thoroughly described in the following sections. In a nutshell, ModelGraft starts with the configuration of the *downscaling and sampling* process (Sec. IV), before entering the *MonteCarlo TTL-based (MC-TTL) simulation* (Sec. V). During the MC-TTL phase, statistics

are computed after a transient period (Sec. V-A), 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. V-B). Once at steady-state, a downscaled number of requests are simulated within a *MC-TTL cycle* (Sec. V-C), at the end of which, the monitored metrics are provided as input to the *self-stabilization* block (Sec. VI): a *consistency check* decides whether to end the simulation (Sec. VI-A), or to go through a T_C *correction* phase (Sec. VI-B), before starting a new simulation cycle.

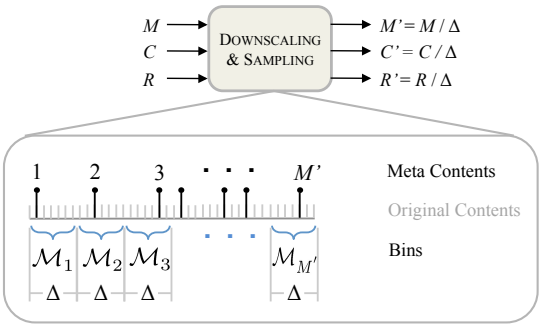


Fig. 3. Downscaling and sampling process.

IV. Downscaling and Sampling

A. Design

The ModelGraft workflow starts with a proper *downscaling* of the original scenario, controlled by the *downscaling factor* $\Delta \gg 1$: specifically, the equivalent TTL-based system has a target cache size $C' = C/\Delta$, a catalog comprising $M' = M/\Delta$ objects, and a target number of simulated requests at steady-state $R' = R/\Delta$. In order to avoid the pitfalls caused by a naïve downscaling process (recall Fig. 1) we need to ensure that the downscaled catalog preserves the main features of the original one, like its popularity distribution. 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. Hence, we denote with Λ the aggregate arrival rate of all objects in the catalog and with $\lambda_n = \Lambda n^{-\alpha} / \sum_{k=1}^{M} k^{-\alpha}$ the rate for the n-th object in the original catalog.

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 $n \in [1, M']$. In ModelGraft, each bin of the original catalog is represented by a single "meta-content" in the downscaled system, i.e., the active catalog comprises M' meta-contents. The key idea is to let each meta-content n to be requested with an *average request rate*, $\bar{\lambda'}_n$ which closely approximates the average request rate of the contents within the respective bin in the original catalog. More formally, for the n-th meta-content, with $n \in [1, M']$, it is required that:

$$\bar{\lambda'}_n = \frac{1}{\Delta} \sum_{m=(n-1)\Delta+1}^{n\Delta} \lambda_m, \tag{6}$$

where the interval $[(n-1)\Delta+1, n\Delta]$ comprises contents of the original catalog that fall within the n-th bin. This design can be achieved by (i) considering M' parallel request generators, i.e., one per each meta-content, each of which is identified by a fixed $n \in [1, M']$; (ii) considering any given meta-content n, varying its instantaneous request rate at each new request, so that its average complies with (6).

B. Implementation

It is easy to see that the simplest implementation of the above requirements boils down to bind the probability of the rate λ'_n at which the n-th meta-content is requested $P(\lambda'_n = \lambda_m)$ with the popularity distribution of the Δ contents inside the respective bin $m \in [(n-1)\Delta + 1, n\Delta]$:

$$P(\lambda'_n = \lambda_m) = \frac{\lambda_m}{\sum_{j=(n-1)\Delta+1}^{n\Delta} \lambda_j} \approx \frac{1}{\Delta} \frac{\lambda_m}{\bar{\lambda'}_n}. \tag{7}$$

While the above requirement (6) is met, a significant downside of this naïve implementation is its space complexity. 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, with an overall memory allocation equal to $M'\Delta = M$ elements, 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 [14], which is an acceptance-rejection method that efficiently generates random variables from a monotone discrete distribution (in this case Zipf distribution) without allocating memory-expensive CDFs, and which is characterized by a $\mathcal{O}(1)$ runtime complexity. Originally proposed in [14] for $\alpha > 1$, this technique has only very recently [2] extended to all non-negative exponents $\alpha > 0$. Recall now that power-laws (and hence Zipf distribution) 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 request rates, at each new request, from a Zipf distribution with exponent α. Indeed, if the request generator associated to the n-th bin, with $n \in [1, M']$, needs to schedule the next request rate for the n-th meta-content, an integer $t \in [1, \Delta]$ is extracted with the aforementioned technique: the relative request rate is, then, computed as $\lambda'_n = \lambda_{(n-1)\Delta+t}$, thus satisfying condition (6). Due to lack of space, we point the interested reader to our technical report [35] for an extensive justification.

V. MC-TTL Simulation

A. Transient

Once the scenario is properly downscaled, and initial uninformed guesses for T_C are provided to ModelGraft, the warm-

up phase of the first MC-TTL *simulation cycle* is started. Given that the duration of the warm-up can be affected by many parameters (e.g., the presence of a conservative cache decision policy, like Leave Copy Probabilistically (LCP) [4], where the reduced content acceptance with respect to Leave Copy Everywhere (LCE) yields much longer transient durations), ModelGraft automatically adapts the duration of the transient period, in order to guarantee the statistical relevance of the monitored KPIs.

B. Steady-state monitor

The convergence of a single node i is monitored via *batch means* on the Coefficient of Variation (CV) of the *measured hit ratio*, $\bar{p}_{hit}(i)$. 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:

$$CV_i = \frac{\sqrt{\frac{1}{W-1}\sum_{j=1}^{W}(p_{hit}(j,i) - \bar{p}_{hit}(i))^2}}{\frac{1}{W}\sum_{j=1}^{W}\bar{p}_{hit}(j,i)} \leq \varepsilon_{CV}, \quad (8)$$

where W is the size of the sample window, and ε_{CV} is a user-defined convergence threshold. 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. 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.

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

$$CV_i \leq \varepsilon_{CV}, \qquad \forall i \in \mathcal{Y}, \quad (9)$$

where $|\mathcal{Y}| = \lceil YN \rceil$ is the *set of the first YN nodes satisfying condition* (8). The rationale behind this choice is to avoid to unnecessarily slow down the convergence of the whole network by requiring condition (8) 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. Although further information about a sensitivity analysis on Y can be found in [35], we anticipate that its typical settings lay in the $Y \in [0.75, 1)$ interval.

C. 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 ModelGraft 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.

VI. Self-stabilization

As described in Sec. III, one of the 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.

A. Consistency check

The *consistency check* is based on the observation that with a downscaling factor Δ, and when $T'_C = T_C$, a TTL cache stores, on average, $C' = C/\Delta$ contents at steady-state. Indeed, adapting (2) to the downscaled scenario (i.e., $M' = M/\Delta$), we have:

$$\mathbb{E}[C'] = \sum_{n=1}^{M'} p_{in}(\lambda'_n, T_C) = C/\Delta. \quad (10)$$

However, unlike LRU caches that have a fixed size (and the oldest content is selected for eviction in LRU fashion), TTL caches have unbounded size (as the old contents remains soft-state in the cache for a fixed TTL, but are not otherwise evicted due to cache size limit). Considering that there exists a strong correlation 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 the controlled variable.

In particular, for each TTL cache we maintain an online average of the number of stored contents as:

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

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 they are geometrically spaced.

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

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

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, supposed to be equal for all the nodes without loss of generality, 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. V-B. 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 measures are reset.

B. T_C correction

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), represents the basis for the controller action. Intuitively, there exists a linear proportionality between C' and T_C: i.e., 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 actual one, and that it needs to be increased in the next step. Viceversa, for a $\widetilde{C} > C'$, the T_C of the correspondent node should be decreased.

As a consequence of the linear 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, the T_C values are corrected, before starting the next one, as:

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

where ${T_C}_i^{(z)}$ 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 VII-C), avoiding at the same time any divergence of the control action (provided that measures on $\widetilde{C}$ are taken at steady-state). This allows ModelGraft to guarantee considerable gains, even when multiple simulation cycles are necessary, due to significantly inaccurate input T_C values.

There is an important condition worth highlighting: i.e., the controller needs to react on measurable quantities (which happens whenever the ratio $C'/\widetilde{C}_i^{(z)}$ can be reliably measured), as opposite to noisy measures (which happens whenever the numerator C' is too small). In particular, this translates into a very simple practical guideline, as it introduces a lower bound to the target cache size of the downscaled system, i.e., $C' = C/\Delta \geq 10$, practically upper bounding the maximum downscaling factor to $\Delta \leq C/10$. For a detailed sensitivity analysis regarding all the parameters involved in the ModelGraft design, we refer the reader to [35].

VII. Validation

Now we validate the ModelGraft engine against classic event-driven simulation, both (i) in the case where we provide *accurate* T_C values as input (to assess the gain of a single MC-TTL cycle), as well as (ii) in the case where we provide *completely wrong* T_C guesses (to assess the self-stabilization capabilities). 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.

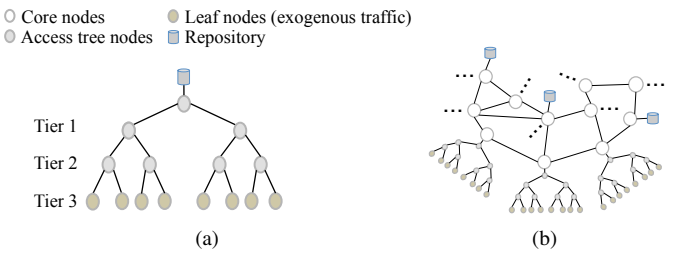


Fig. 4. Network Topologies: (a) 4-level binary tree, (b) CDN-like.

A. Very-large Scale Scenario

To assess ModelGraft accuracy, we consider the largest scenario we can investigate via event-driven simulation gathered via ccnSim, already shown to be among the most scalable ICN software tools [34]. To stretch the boundaries reachable by event-driven simulation, we integrate the rejection inversion sampling – to the best of our knowledge, this represents *the first performance evaluation of ICN networks with content catalogs in the order of billions*.

The validation scenario represents an ICN access tree network [24], where the topology is a N=15-nodes 4-level binary tree depicted in Fig. 4(a). A single repository, connected to the root node, stores a $M = 10^9$ object catalog, where objects have Zipf popularity distribution with exponent $\alpha = 1$. An overall $R = 10^9$ requests, following an Poisson IRM process, are injected at each leaf nodes, at a rate of $\Lambda = 20$ req/s per leaf.

The cache size of each node is fixed at $C = 10^6$, resulting in a cache to catalog ratio of $C/M = 0.01\%$. Three different cache decision policies are considered for the comparison: (i) LCE, where fetched contents are always cached in every traversed node; (ii) LCP(1/10), that probabilistically admits content in the cache (configured so that one over ten fetched contents is cached on average); (iii) 2-LRU [16, 22], 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 hit event in the first cache.

According to our rule of thumb $C' = C/\Delta \geq 10$, the maximum downscaling factor is $\Delta = 10^5$. Additionally, equations (9) 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 iterate MC-TTL simulations until the measured average cache size of those nodes is within 10% of the expected size $C' = C/\Delta$. It is worth stressing that while we cannot report a thorough sensitivity of the above parameters in this paper due to size limits, an extensive account of their (limited) impact is available to the interested reader at [35].

TABLE I
T_C 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%

TABLE II
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}	CPU	Gain	Mem [MB]	Gain
LCE	Simulation	33.2%	11.4 h	**194x**	6371	**168x**
	ModelGraft	**31.4%**	**211 s**		**38**	
LCP(1/10)	Simulation	35.4%	7.3 h	**90x**	6404	**168x**
	ModelGraft	**34.0%**	**291 s**		**38**	
2-LRU	Simulation	37.0%	10.8 h	**97x**	8894	**234x**
	ModelGraft	**36.1%**	**402 s**		**38**	

B. Accurate initial T_C

To illustrate values taken by the characteristic time T_C in this large scale scenario, and its dependency on the cache decision policy, Tab. I reports T_C measured by event-driven simulation at different depths of the tree. It clearly appears that the cache decision policy has the largest impact on the T_C values (even larger than topological position in the network). Specifically, the more conservative the policy (e.g., LCP or 2-LRU), the larger the T_C values, which can vary up to one order of magnitude among different policies. Intuitively, with caching policies that are more conservative in admitting new contents, (popular) stored contents are cached for longer time (which increases the overall hit probability).

To validate the ModelGraft workflow, we start with a single MC-TTL cycle: to do so, we feed ModelGraft with accurate T_C estimates, so that no iteration is necessary. Results are reported in Tab. II, where mean values of 10 different runs for three KPIs are reported: mean hit ratio p_{hit}, CPU time, and memory occupancy (for which relative gains are also highlighted). Results are noteworthy in that (i) the discrepancy between p_{hit} measured by event-driven vs ModelGraft remains always under 2%, (ii) ModelGraft achieves significant gains, of about two orders of magnitude, for both CPU time and memory occupancy. Additionally, it is interesting to notice that there is neither trace of the (iii) accuracy/speed trade-off, as one could typically expect [27], nor trace of (iv) memory/CPU tradeoff [13], i.e., cases where an algorithm either trades increased space (e.g., cached results) for decreased execution time (i.e., avoid computation), or viceversa.

C. Inaccurate initial T_C

For an arbitrary scenario, T_C is however unknown: it is, thus, important to assess the performance of ModelGraft when non-informed guesses are provided as input. Denote with $T_C^0(i)$ the initial characteristic time for node i, and with $T_C^{sim}(i)$ the accurate estimate of the characteristic time gathered via simulation.

To purposely introduce errors in a controlled fashion, the input characteristic time of each node is set as $T_C^0(i) = b(i)T_C^{sim}(i)$, where the *multiplicative factor* $b(i) \in [1/(Bu), Bu]$ is obtained by multiplying a bias value $B \in [1, 100]$ (equal for all the nodes), by a uniform random variable $u \in (0, 1]$, in case of both overestimation ($b(i) > 1$) and underestimation ($b(i) < 1$). Notice that in case of maximum bias (i.e., $max|B| = 100$), $T_C^0(i)$ will differ from $T_C^{sim}(i)$ by up to two orders of magnitude, and it will differ for each node in the network due to the uniform variable u.

Results are reported in Fig. 5 as a function of the multiplicative factor b, depicting CPU and memory gain (ratio of event-driven over ModelGraft), vs accuracy loss (absolute error of p_{hit}) and number of MC-TTL simulation cycles before convergence. Several remarks follow. First, (i) p_{hit} accuracy loss is not affected by the magnitude of the bias of the initial guess: this remarkably confirms ModelGraft to converge to the accurate system performance even for very harsh estimation of the initial T_C values, making the methodology very robust for practical purposes. Additionally, (ii) even though the number of cycles needed to reach such accuracy increases for increasing overestimation/underestimation bias, it, however remains bounded by a small number in practice. It follows that, (iii) while CPU gain is maximum in the absence of bias (about 200x, when no iterations are required), the benefits brought by ModelGraft are, however, still significant (above 50x-100x) even for very large T_C biases (100x overestimation - 1/100x underestimation, respectively). Finally, (iv) memory gain is, as expected, independent of the initial T_C bias.

VIII. RESULTS

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.

A. Internet-scale Scenarios

We now aim at investigating Internet-scale scenarios, whose content catalogs are estimated [26] to be in the order of $\mathcal{O}(10^{11})$ – $\mathcal{O}(10^{12})$, i.e., *two orders of magnitude larger than those considered in the previous section.*

We consider two scenarios: the ICN-like one, depicted in Fig. 4(a), which models 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 $\Delta = 10^5$. The second scenario, depicted in Fig. 4(b), models 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

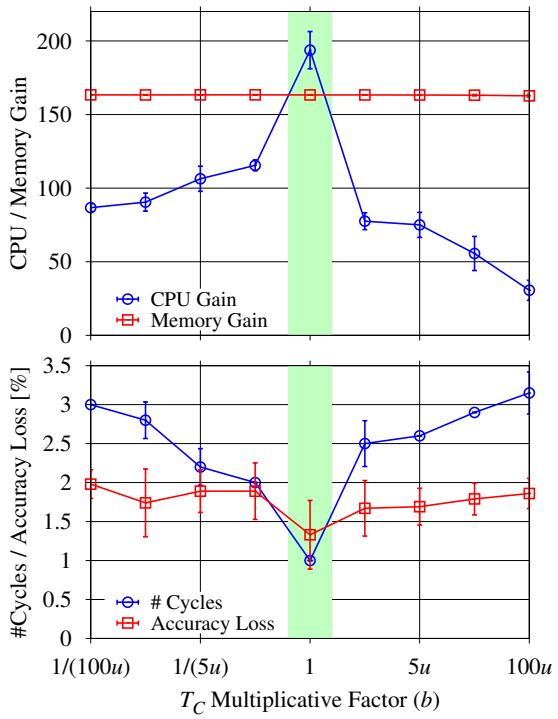


Fig. 5. T_C sensitivity - ModelGraft performance in very large scenario: 4-level binary tree, LCE cache decision policy, $M = 10^9, R = 10^9, C = 10^6, \Delta = 10^5$, and variable input T_C values.

further attached to each backbone node. In this scenario, we let the cache size be $C = 10^7$, which allows to increase the downscaling to $\Delta = 10^6$.

As before, we set $Y = 0.75$, $\varepsilon_{CV} = 5 \cdot 10^{-3}$, and $\varepsilon_C = 0.1$ and run experiments on the same Intel Xeon E5-1620, 3.60GHz, with 32GB of RAM. Clearly, 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). At the same time, we can estimate the expected memory occupancy and CPU times by fitting and cross-validating several scenarios with simple models. Despite related to the very specific implementation of the ccnSim simulator, estimates are, however, useful to project ModelGraft gains. While a thorough analysis is provided in [35] due to lack of space, general trends are discussed herein: on the one hand, the event-driven approach of ccnSim has a $\mathcal{O}(M) + \mathcal{O}(NC)$ memory cost (being N the number of caches in the network), meaning that the allocation of memory space is mostly influenced by the cardinality of the catalog, rather than by the cache size (being $M \gg C$). On the other hand, the CPU time varies linearly with the number of total requests, i.e., $\mathcal{O}(R)$.

B. Projected gains

Results for the Internet-scale scenarios are shown in Tab. III, which reports the mean values of p_{hit}, CPU time, and memory usage, along with the number of MC-TTL cycles gathered via ModelGraft, and estimates of CPU time and memory occupancy for the classic event-driven approach.

TABLE III
INTERNET-SCALE SCENARIOS: MODELGRAFT RESULTS AND PROJECTED GAINS VS EVENT-DRIVEN SIMULATION.

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

Consider the ICN-like scenario first. Notice that, whereas memory requirements for event-driven simulation increased with respect to scenarios in the previous section (due to the need of mapping seed copies with respective repositories), memory usage in ModelGraft increases only slightly (since the mapping scales as M'): consequently, memory gain increases significantly. Second, notice that CPU gains maintain to 270×, which happens since in this case our initial guess of T_C was accurate enough to let ModelGraft converge after one cycle.

For the CDN-like scenario, instead, memory gain increases by one further order of magnitude, in reason of the larger downscaling factor $\Delta = 10^6$. Conversely, CPU time gain reduces due to the larger number of cycles needed to end the simulation: specifically, given that our initial T_C guess was not accurate enough, ModelGraft took three cycles to converge, reducing gains of the techniques to a still very significant 96×.

IX. Related Work

Hybrid approaches have been considered to make it practical to study large scale networks even with commodity hardware, and at a reasonable time scale. 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 more closer communication networks, in the forms of large scale IP networks [30, 21], wireless sensor networks [20], and control theory [5].

What presented in this paper finds a strict correspondence with the work in [30], where large IP networks are scaled to reduce the computational requirements of simulations and simplify performance prediction. The idea consists in feeding a suitable scaled version of the system with a sample of the input traffic, while changing the scaling rule according to the type of TCP/UDP flows traversing the network.

TCP networks are also considered in [21], where the authors propose a scalable model which is easily comparable with discrete event simulators due to its time-stepped nature. In particular, by refining a known analytical model [29] based on ordinary differential equations, they show that their approach yields accurate results with respect to those of the original networks, and that, 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 [20], or the steady state throughput of TCP/IP networks [30, 21]. Our methodology is not only novel, but also practical, as it is fully integrated in open-source tools [1].

X. Conclusion

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

ModelGraft features, like decoupled dynamics of content requests and the use of TTL caches, pave the way for a further improvement, which we keep as future work: by resorting to a map-reduce approach, different cores could simulate the request process for different parts of the downscaled catalog (i.e., bins can be assigned in order to equalize to load of each core); then, once a consistency check is needed at the end of a MC-TTL cycle, measurements about the actual cache occupancy, coming from all the cores, can be linearly aggregated (i.e., by simply summing them) in order to check for (12). In case a further cycle would be needed, the T_C will be corrected, and the different cores will restart to simulate their part of the catalog. Since there would be no need of message passing between the cores during the MC-TTL simulation, this approach could remarkably reduce the CPU time, especially if projected in a multi-core cluster environment.

Acknowledgments

This work has been carried out at LINCS (http://www.lincs.fr). 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] ccnSim Simulator. http://perso.telecom-paristech.fr/~drossi/ccnSim.

[2] Zipf distributed random number generator . https://github.com/apache/commons-math/blob/master/src/main/java/org/apache/commons/math4/distribution/ZipfDistribution.java.

[3] 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*, PP(99):1, 2015.

[4] S. Arianfar and P. Nikander. Packet-level Caching for Information-centric Networking. In *ACM SIGCOMM, ReArch Workshop*. 2010.

[5] 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.

[6] M. Cha, H. Kwak, et al. I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system. In *ACM IMC*. 2007.

[7] H. Che, Z. Wang, et al. Analysis and design of hierarchical web caching systems. In *Proc of IEEE INFOCOM*. 2001.

[8] A. Dan and D. Towsley. An Approximate Analysis of the LRU and FIFO Buffer Replacement Schemes. *SIGMETRICS Perform. Eval. Rev.*, 18(1):143, 1990.

[9] N. Fofack, P. Nain, et al. Performance evaluation of hierarchical TTL-based cache networks. *Elsevier Computer Networks*, 65:212 , 2014.

[10] C. Fricker, P. Robert, et al. A versatile and accurate approximation for LRU cache performance. In *Proc. of ITC 24*. 2012.

[11] 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.

[12] 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.

[13] M. E. Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401, 1980.

[14] W. Hörmann and G. Derflinger. Rejection-inversion to generate variates from monotone discrete distributions. *ACM Trans. Model. Comput. Simul.*, 6(3):169, 1996.

[15] J. Jaeyeon, A. W. Berger, et al. Modeling TTL-based Internet caches. In *Proc. of IEEE INFOCOM*. 2003.

[16] T. Johnson, D. Shasha, et al. 2q: A low overhead high performance buffer management replacement algorithm. In *20th International Conference on Very Large Data Bases (VLDB)*, pages 439–450. 1994.

[17] W. King. Analysis of paging algorithms. In *IFIP Congress*. 1971.

[18] N. Laoutaris, H. Che, et al. The LCD interconnection of LRU caches and its analysis. *Performance Evaluation*, 63(7), 2006.

[19] E. Leonardi and G. Torrisi. Least Recently Used caches under the Shot Noise Model. In *Proc. of IEEE Infocom*. 2015.

[20] P. Levis, N. Lee, et al. TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. In *Proc. of ACM SenSys*. 2003.

[21] 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.

[22] V. Martina, M. Garetto, et al. A unified approach to the performance analysis of caching systems. In *Proc. of IEEE INFOCOM*. 2014.

[23] D. Berger et al. Exact Analysis of TTL Cache Networks: The Case of Caching Policies Driven by Stopping Times. In *Proc. of ACM SIGMETRICS Conference*, pages 595–596. 2014.

[24] S. Fayazbakhsh et al. Less Pain, Most of the Gain: Incrementally Deployable ICN. *SIGCOMM Comput. Commun. Rev.*, 43(4):147, 2013.

[25] N. Fofack et al. On the performance of general cache networks. In *Proc. of VALUETOOLS Conference*, pages 106–113. 2014.

[26] K. Pentikousis et al. Information-centric networking: Evaluation methodology. Internet Draft, https://datatracker.ietf.org/doc/draft-irtf-icnrg-evaluation-methodology/, 2015.

[27] M. Rosenblum et al. Complete Computer System Simulation: The SimOS Approach. *IEEE Parallel Distrib. Technol.*, 3(4):34, 1995.

[28] G. Xylomenos et al. A survey of information-centric networking research. *Comm. Surveys and Tutorials, IEEE*, 16(2):1024, 2014.

[29] V. Misra, W. Gong, et al. Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED. *SIGCOMM Comput. Commun. Rev.*, 30(4):151, 2000.

[30] 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.

[31] E. Rosensweig, D. Menasche, et al. On the steady-state of cache networks. In *Proc. of IEEE INFOCOM*. 2013.

[32] E. J. Rosensweig, J. Kurose, et al. Approximate Models for General Cache Networks. *IEEE INFOCOM*, pages 1–9, 2010.

[33] G. Rossini and D. Rossi. Coupling caching and forwarding: Benefits, analysis, and implementation. In *Proc. of ACM SIGCOMM ICN*. 2014.

[34] M. Tortelli, D. Rossi, et al. ICN software tools: survey and cross-comparison. *Elsevier Simulation Modelling Practice and Theory (SIMPAT)*, 63:23, 2016.

[35] M. Tortelli, D. Rossi, et al. Modelgraft: Accurate, scalable, and flexible performance evaluation of general cache networks. Telecom ParisTech Tech. Rep., http://www.enst.fr/~drossi/paper/ModelGraft.pdf, 2016.