FOCAL: Forwarding and Caching with Latency awareness in Information-Centric Networking

Giovanna Carofiglio*, Leonce Mekinda†, Luca Muscariello†
* Cisco Systems, † Orange Labs Networks,
gcarofig@cisco.com, firstname.lastname@orange.com

***Abstract*—Latency minimization is an important network optimization criterion which becomes an even more compelling feature in 5G networks. Information-Centric Networking (ICN) appears a promising candidate technology for building an agile communication model that reduces latency via a fully distributed and adaptive delivery approach coupling in-network caching and forwarding. In the paper, we investigate the role of latency awareness on ICN delivery performance and introduce FOCAL, an approach combining novel caching and forwarding strategies to jointly reduce end-user experienced latency with no network signaling nor coordination between routers.**

FOCAL gathers a latency-proportional probabilistic caching policy, with a load-aware dynamic forwarding strategy, that preferentially routes popular content requests through a single path (set of caches), while globally achieving minimum network load and user content delivery time, thus delay minimization. By means of ICN simulations, we assess the superiority of FOCAL over existing alternatives given by the combinations of known caching policies and forwarding strategies: It results a reduced end-user delivery performance coupled with faster convergence to average/variance figure and higher self-adaptiveness to varying traffic/network conditions.

I. Introduction

If latency minimization is already an important traffic engineering criterion in current networks, it is anticipated as a founding principle for the architectural design of 5G networks. An efficient orchestration of edge caching, on-the-fly processing and traffic load-balancing appears essential to relieve congestion and to accommodate QoE (Quality of Experience) requirements of latency-sensitive applications, like pervasive video or tactile Internet.

Research on ICN (Information-Centric Networking), has recently highlighted the benefits of content-centric over host-centric communication in terms efficient data delivery, but also optimized use of network resources, simplified management of mobility and embedded security.

In ICN a tight coupling exists by definition between distributed forwarding and caching operations: the use of hop-by-hop dynamic forwarding determines the arrival process at in-network caches, where the persistence of content can be locally optimized. To close the loop, the resulting hit/miss performance affects link loads and forwarding decisions to achieve overall performance optimization. Therefore, latency reduction can be achieved, in ICN, via both in-network caching and hop-by-hop distributed forwarding. The goal of this paper is to investigate the impact of latency-awareness on caching decisions alone, then to study the interaction with different dynamic forwarding strategies and to propose a combined approach. Previous study focused on the definition of ICN caching and forwarding strategies with the aim of minimizing a network cost function, very few considering latency. In this paper, we investigate for the first time the role of latency awareness on ICN data delivery performance under dynamic bandwidth sharing and network congestion.

More precisely, we build upon an initial proposal of Latency-Aware Caching (LAC) and enhance it to strengthen latency dependency in probabilistic caching decisions (LAC+). If the benefits brought by LAC+ are clear on a single cache or a system of caches working under random request forwarding, the interaction with smart forwarding strategies is the second step for the definition of FOCAL. We consider as starting point the optimal load-balancing (LB) solution derived in [2] to achieve maximum load minimization by distributed and dynamic monitoring of the residual round trip time behind output interfaces. Intuitively, such content-agnostic fine-granular forwarding strategy does not help differentiate the arrival process at caches along different paths and hence realize implicit latency-aware cache coordination. To this aim, we introduce a novel load-balancing strategy, *LB-Perf* that locally monitors more popular content requests and persists in routing them through a single path, while applying the agnostic LB approach to the aggregate of less popular requests. The performance of FOCAL, given by the combination of LAC+ and LB-Perf, is further evaluated by means of ICN simulations in various network scenarios, to show the benefits w.r.t existing alternatives, namely combinations of known caching policies (LRU, probabilistic caching, Leave-a-Copy-Down) and forwarding strategies (random, load-balancing, load-balancing with persistent forwarding). Promising results are obtained in terms of reduced end-user delivery performance coupled with faster convergence to average/variance figure and higher self-adaptiveness to varying traffic/network conditions.

The remainder of the paper is organized as follows. Sec. II describes related work. In Sec.III we introduce FOCAL by presenting latency-aware caching and forwarding strategies in this order. The evaluation results are gathered in Sec.IV, while Sec.V concludes the paper.

II. Related work

In the context of ICN research, we identify two categories of related work: proposals introducing enhancements of classical cache management policies for a given cost function and

studies focusing on forwarding strategies to optimize request-to-cache routing.

Latency-aware caching: Within the panoply of cache management proposals, some leverage content placement (e.g. [15], [8]) while others deal with caching mechanisms based on selective insertion and replacement in cache (e.g. [11], [1], [6], [10]). The first class of approaches is appropriate for small-scale controlled environments like a CDN (Content Delivery Network), where topology and content catalog are known a priori. Either [15] and [8] deals with video streaming in ICN and orchestrate caching and scheduling of requests to caches in order to create a cluster of caches with a number of guaranteed replicas. Unlike these approaches, our previous ([3]) and current work on latency-aware caching belong to the second class of caching solutions by defining a decentralized solution that automatically adapts to changes in content popularity, network variations etc. by leveraging content insertion in cache. We share the same objective as in [1], where authors propose a congestion-aware caching mechanism for ICN, based on estimation of local congestion, of popularity and of bottleneck position. Differently from our work, their congestion estimate does not differentiate content items in terms of latency. Similar considerations hold for other related approaches: the ProbCache work in [11], using the same cache probability for every content item at a given node and the cooperative caching mechanism in [6]-[10] exploiting overall popularity and distance-to-server.

Caching and Forwarding Interaction: The search for an optimal interplay between in-network caching and forwarding has drained effort in ICN research, as driven by different user or network performance objectives. The optimal cache placement and forwarding problem applied to ICN was tackled in [13]. Unlike our approach to forwarding, bandwidth sharing is not taken into account by such formulation, leading to results that are more appropriate for network dimensioning purposes than for end-user latency minimization. Between the contributions that focus on cache-aware forwarding strategies, [4] proposes an Interests-to-neighbor forwarding aiming at maximizing a difference of potentials, whose strength decreases with the distance to the content location. Similarly, in [12], authors suggest to forward Interests to the neighboring node that advertised the highest hit probability for the requested content object. A closer work to ours is [14], where an optimization framework is defined to jointly handle backpressure-based forwarding and LFU-like content placement. The work designs a control plane that feeds the actual chunk-level data plane with flow rates and queue sizes in order to operate optimal content placement and request forwarding. Nodes must update their neighbors about their own queue states. Our solution differs in that it does not require signaling and does fully-distributed and dynamic cache insertion/replacement without requiring optimal content placement a priori. The latter aspect is important to guarantee self-adaptiveness to varying network/traffic conditions.

III. FOCAL

A. Latency-aware caching strategies

Taking into account latency awareness into cache management, can improve alone the delivery time of latency-sensitive applications and on average the global delivery time perceived by user, as shown in [3]. Before considering the joint effect of latency-aware caching and forwarding, we present a novel stochastic caching mechanism, exploiting monitored latency information for cache insertion decisions and not involving cache coordination. The novel latency-aware caching policy is named *LAC+* and builds upon the LAC proposal in [3] that we summarize below. The enhancement of *LAC+* consists in strengthening latency-dependency in probabilistic cache decisions w.r.t. *LAC*, based on an online monitoring and estimation of the second order moment of latency distribution.

*1) **LAC**:* In ICN, when a requested content object is not available in cache, a cache miss event occurs and the Interest is forwarded up to the first hitting cache where the corresponding Data is retrieved and sent downstream to the client. On the reverse path to the client, every cache decides whether or not storing the object, at the cost of triggering the eviction of another object due to finite storage space constraints.

According to LAC, at time t, the decision to store object k is positive with a given probability $p_k(t)$ and negative otherwise (with probability $1 - p_k(t)$). As a special case of a prior proposal from [3], we characterize $p_k(t)$ as

$$p_k(t) \equiv \min\left(\epsilon\frac{T_k(t)}{\bar{T}(t)}, 1\right) \tag{1}$$

where $T_k(t)$ refers to content k monitored latency at time t, $\bar{T}_k(t)$, $\bar{T}(t)$ respectively to the temporal averages for content k and for all cached contents computed up to time t. $\bar{T}(t)$ and $\bar{T}_k(t)$ are estimated using Exponentially Weighted Moving Averages (EWMA), with a weight associated to the historical value of average latency set to $\alpha = 0.9$. The cache insertion probability, $p_k(t)$ results from the product of a small factor, ϵ, modulated by the ratio of its retrieval latency over the average latency of cached objects. A first assessment of *LAC* performance suggested that further benefits may result from strengthening the latency-awareness contribution by highlighting second order moment characteristics of monitored latency. This is the rationale behind *LAC+* proposal.

*2) **LAC+**:* *LAC* may suffer from the slow convergence of any other probabilistic approach (see e.g. [11]), due to the small ϵ factor. In simulations, we observe a non negligible time for even very popular objects to be persistently cached. We recall that the ideal behavior of a cache should be to capture the most valuable objects (according to a defined cost function), while avoiding unnecessary replication across network of caches. Unnecessary replication is typically object replication below a bottleneck or the lack of implicit coordination between neighboring caches. *LAC+* achieves such objective by supplementing LAC with an outlier tracking function, meant to estimate second order moment of observed latency distribution. The outlier tracking function, denoted as

Θ_k, significantly increases cache insertion probability for those exhibiting an exceptionally high deviation in comparison the other content objects. Such outliers correspond to significantly higher-than-average latency items, that is important to cache even when not very popular. According to *LAC+*, at time t, the decision to store object k is positive with a given probability $p_k^+(t)$ and negative otherwise (with probability $1-p_k^+(t)$). We define $p_k^+(t)$ as the linear combination of two terms:

$$p_k^+(t) \equiv p_k(t) + (1 - p_k(t))\Theta_k(t) \quad (2)$$

Let μ_t and σ_t be the average and standard deviation of all $\bar{T}_i(t), \forall i \in \mathcal{K}$, at a given node. The n^{th} quantile being $Q_n(t) = \mu_t + n\sigma_t$, it follows that

$$\Theta_k(t) \equiv \max\left(\frac{\bar{T}_k(t) - Q_n(t)}{\bigvee_{i\in\mathcal{K}} \bar{T}_i(t) - Q_n(t)}, 0\right) \quad (3)$$

$p_k^+(t)$ inherits its first term from *LAC*. Its added value dwells in the second term, that allows to account for objects with a sensibly higher-than-average latency, in order to cache them even when not very popular (namely, when not selected by the filtering embedded in the first term). Its purpose is to strengthen the latency dependency of the caching decision to favor, by means of Θ_k, a positive caching decision for those objects whose retrieval can be very costly: e.g. long distance to the hitting cache, upstream congestion or severe bandwidth limitations. $\Theta_k(t)$ is defined as the probability that object k average latency at time t is an outlier.

B. Latency-aware Forwarding strategies

Latency reduction can be also achieved via smart hop-by-hop request forwarding strategies trying to minimize *i)* distance to the first hitting cache, *ii)* congestion status of the network. To such extent, the presence of multiple paths is clearly essential. Using as baseline for comparison the *uniform random forwarding* approach that blindly selects with equal probability output interfaces in FIB, our focus is on the family of distributed, dynamic load-balancing approaches whose objective is to split content requests over time and through the available output interfaces such as to minimize *i)-ii)* on average. In [2], a load balancing scheme is derived from a joint optimization of end-user rate/congestion control and multipath forwarding under the objective of minimizing the maximum link load network-wide. The minimization of the maximum link load implicitly leads to a sensible reduction of the overall average latency as it can be appreciated in the simulated scenarii. Hereinafter we refer to such approach simply as *Load balancing (LB)*. *LB* selects available output interfaces per FIB entry randomly according to computed weights. At the beginning, each interface has the same weight equal to one and the randomized forwarding process is uniform over available output interfaces. This allows to probe all available interfaces and to monitor the average number of outstanding Pending Interests (PI) per FIB entry and per interface. Such metric reflects the residual latency due to first hitting cache distance and congestion status. After such initial phase, the computation of the weights driving interface selection simply consists in taking the average number of PI per FIB entry and per output interface normalized over the total average number of PI per FIB entry (so that weights are comprised between 0 and 1). Ideally, LB works on per-content FIB entries enabling a fine granular load-balancing at flow scale. However, a feasible approximation that keeps limited FIB state replaces per-content with per-prefix entries aggregating all content names behind the same prefix (FIB lookup is assumed to be Longest-Prefix Match). Note that in our simulations we adopt a per-content LB approach with the objective to quantify its best performance.

Algorithm 1: The most popular content items are sampled in PopularFiles. Create flow bundles, one per FIB entry, based on observed interest volume and associate persistent faces to popular content items.

```
At update time (every ΔT);
Faces are ranked every ΔT_f > ΔT;
T+ = ΔT ;
IsPersistentDisabled = FALSE;
foreach FileName in PopularFiles do
    prefix = GetFIBPrefix(FileName) ;
    OuputFaces = GetOutputFaces(prefix) ;
    FlowBundle = GetFlowBundle(prefix) ;
    Face = OuputFaces.Begin();
    Sort(FlowBundle by InterestCounter) ;
    if ( T >= ΔT_f ) then
        Sort(FaceRecord by weight) ;
        T = 0 ;
    end
    CumSum = 0 ;
    foreach ( FlowRecord in FlowBundle ) do
        CumSum + = FlowRecord.InterestCounter ;
        Weight = CumSum/FlowBundle.Norm ;
        while ( Face != OuputFaces.End() ) do
            if ( Weight < Face.weight ) then
                FlowRecord.SetFace(Face);
                break ;
            else
                CumSum = FlowRecord.InterestCounter ;
                Weight = CumSum/FlowBundle.Norm ;
                Face = OuputFaces.Next();
            end
        end
    end
end
```

LB may achieve significant improvement of overall end-user throughput/latency over uniform random forwarding via load-aware utilization of multiple paths. However, it is not capable of realizing implicit cache coordination for caches along different paths, as a consequence of its randomized weighted split, that load-balance Interest for the same content over all output interfaces according to the weights. To understand this issue, let us consider the case of three output interfaces available for a given content k with associated weights, w_1, w_2, w_3. At each incoming request for content k, *LB* splits the Interest arrival process over time into three output processes, with rate respectively w_1, w_2, w_3 of the total, without selecting the same output interface for a given chunk request. As a result, the

Algorithm 2: Popularity based persistent face selection.

```
At Interest I arrival with name /p/file_name/chunk_name ;
I matches name prefix /p in the FIB ;
OuputFaces = GetOutputFaces(prefix = /p) ;
if ( IsPersistentDisabled ) then
    LoadBalancing.Update(OuputFaces.weights) ;
    FaceID = LoadBalancing.SelectFrom(OuputFaces) ;
else
    PopularitySampler.Insert(I) ;
    if (PopularitySampler.Find(FileName(I))) then
        PopularFiles.ManageHit(I) ;
    end
    if (PopularFiles.IsPopular(FileName(I)) ) then
        FlowRecord = FlowBundle.Find(FileName(I)) ;
        FlowRecord.InterestCounter++ ;
        FlowBundle.Norm++ ;
        FaceID = FlowRecord.GetFace() ;
        if ( FaceID isEmpty ) then
            FaceID = LoadBalancing.GetFace(OuputFaces) ;
        end
    else
        FaceID = LoadBalancing.GetFace(OuputFaces) ;
    end
end
DoSendInterest(I, FaceID) ;
function ManageHit(Interest = I)
    prefix = GetFIBPrefix(FileName(I)) ;
    FlowBundle = GetFlowBundle(prefix) ;
    if PopularitySampler.IsPopular(Filename(I)) then
        FlowRecord = FlowBundle.Find(FileName(I)) ;
        if FlowRecord isEmpty then
            FlowBundle.Insert(FileName(I)) ;
        end
    end
end
```

arrival process at caches along the three paths has the same characteristics (except for the rate) of the original one, leading to caches operating independently and storing the same items.

Intuitively such behavior advantages most popular objects cached with high probability over all available paths, but reduces overall caching benefits due to lack of cache coordination. Instead, splitting Interests in a way to persist the selection of one or few single output interfaces over time on a per-content basis, (while keeping per-prefix load-balancing according to *LB* weights) would differentiate the arrival process at caches along the three paths, so realizing implicit cache coordination and better overall performance. Such idea inspires our proposal for an enhanced load balancing scheme, that we name *LB-Perf* (Load Balancing with Persistent Forwarding).

In an initial phase, *LB-Perf* computes per-prefix weights to associate to available output interfaces as in *LB* case. FOCAL is also equipped by a popularity sampler which continuously monitors the most popular objects and store their name locally. The method to perform online popularity estimation is out of scope of this paper, but in our simulation we have used the a k-LRU filter [9]. In our simulations, we set each sub k-LRU cache equal to 40 objects (in the first simple scenario) or to 160 (in the other scenarii). Objects found in the last sub-cache are considered high popularity and get every of their chunk hit counted for precise flow sizing. In a more general implementation of our mechanism, we do not suggest to use k-LRU which, while being simple, requires k to be very large when popularity is measured in terms of observed traffic and not in terms of number of content item requests. However a content item (object or file) request is difficult to be identified in practice, as different clients can request the same object using distinct permutation of the chunks sequence numbers.

For a given prefix, the samples most popular items are grouped into flow bundles as reported in Algorithm 1. Each bundle contains items with consecutive popularity up to the face weight, hence size of each bundle depends on the face weight as reported in Algorithm 1. Thus, for more popular items a single output interface is persistently selected by selecting less congested interfaces (with higher weights) first. For all other items, face selection obeys to standard *LB* rule, see Algorithm 2. Every ΔT seconds, the most popular items are reassigned to flow bundles according to face weights which are, on the other hand, updated independently.

IV. Evaluation

In this section we assess the performance of FOCAL by means of ICN simulations in three different scenarii and against existing forwarding and caching alternatives given by the combination of the following known caching policies:

- *LRU*, Least Recently Used: deterministic cache insertion of every item arriving at the cache coupled with LRU replacement,
- ϵ-*LCP*, Leave a Copy Probabilistically: a probabilistic cache insertion with probability ϵ ($\epsilon = 10^{-3}$ where not specified) coupled with LRU replacement,
- *LCD*, Leave a Copy Down: a content object retrieved at $l-th$ cache along a path is cached at $(l-1)$-th cache only, rather that in all caches from 1 to $l-1$ ([7]),

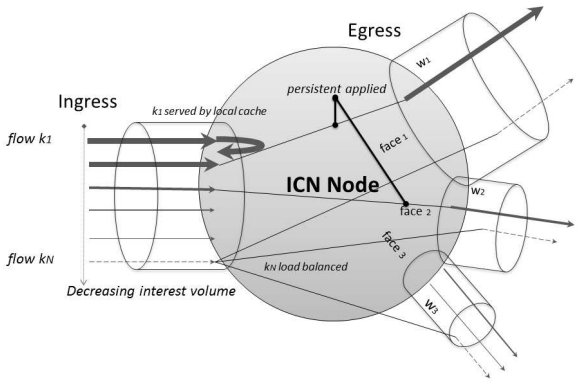
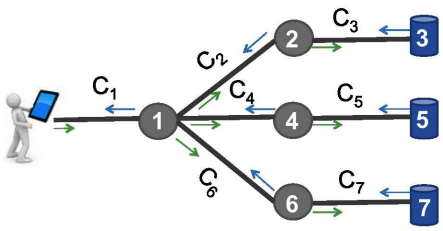
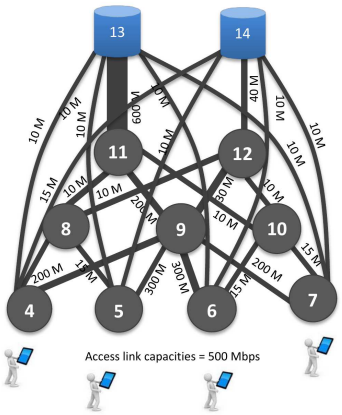


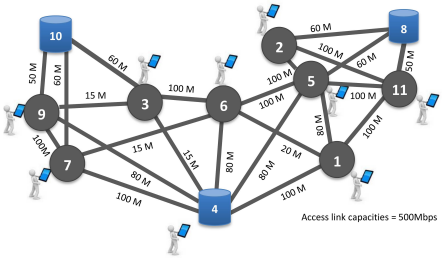
Fig. 1. The face selection algorithm is depicted: Load balancing with persistent face selection for popular content.



(a) Linear topology with forwarding branches.



(b) Fat tree with direct links to repositories.



(c) Abilene-like topology.

Fig. 2. Network topologies used in the evaluation.

- *LAC*, Latency-Aware Caching: the approach proposed in [3],
- *LAC+*, enhanced LAC: our approach presented in Sec.III-A2

and forwarding strategies:

- *Uniform*: uniform selection performed on nearly per-packet basis of output interfaces stored in FIB entries aggregated per-prefix;
- *LB*, Load-Balancing: load-aware selection performed on per-packet basis of output interfaces stored in FIB entries aggregated per-prefix, as in [2].
- *LB-Perf*, Load-Balancing with Persistent forwarding: our approach presented in Sec.III-B. The approach combining LB-Perf with LAC+ caching is denoted as FOCAL.

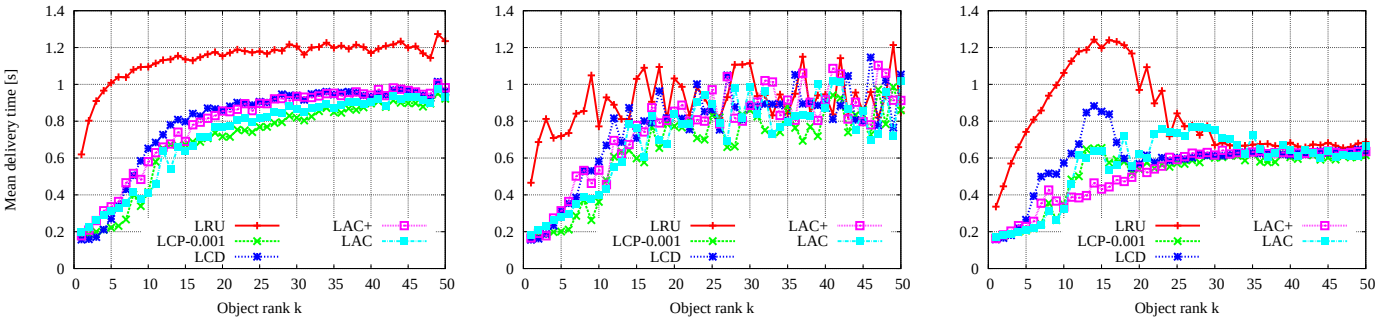
To this purpose, we implement FOCAL and its alternatives in the packet-level NDN simulator CCNPL-Sim (http://systemx.enst.fr/ccnpl-sim). Three topologies are considered: a linear topology with forwarding branches, a hierarchical fat tree with direct access to content repositories and a non-hierarchical meshed topology (Abilene-like).

A. Linear topology with forwarding branches

We simulate the simple topology in Fig.2(a) to show: *i)* the improvement of latency-aware caching policies in presence of random forwarding, i.e. without latency-awareness in link selection; *ii)* the interaction with forwarding strategies and overall superiority of FOCAL. The tests consist in a branched network of ICN nodes capable of storing up to 10 content items per cache. We simulate a 55-hour traffic involving a single content producer that serves a catalog of 20,000 objects. Popularity is Zipf-like distributed with parameter $\alpha = 0.9$. It implies that the ten most popular items weight 19% of the traffic. Each content item is conveyed in chunks of 3kB and has a total size of 2MB (the same size is used in all scenarios presented in the paper). In all simulations reported in the paper, data retrieval is managed by an implementation of the transport protocol presented in [2]. Link capacities are limited to C_1=600Mbps, C_2=60Mbps, C_3=20Mbps, C_4=100Mbps, C_5=30Mbps, C_6=300Mbps, C_7=50Mbps. The object request process feeding node 1 is assumed to be Poisson with rate parameter $\lambda = 3$ objects/s. The maximum long term link load in the network does not exceed 50% of utilization. In Fig.4(a), we neglect the impact of forwarding, by considering a uniform selection of the three output interfaces at node 1. All caching policies aim in a more or less effective way at caching the most popular items on the first cache, the following ones in terms of more popular items at the second level of caches (2,4,6). Except for LRU which under-performs, the other policies show similar results. Clearly, the second level caches work independently under uniform forwarding, because they receive the same arrival process sampled at $1/3$ of the total request rate. This still happens in presence of LB forwarding (Fig.4(b)) due to the blind load-aware forwarding of packets over the three interfaces. The benefits in terms of latency reduction deriving from load distribution across the three paths appear to be negligible compared to the lack of implicit cache coordination. A significant reduction up to 40% in average delivery time over the entire catalog can

Forwarding / Caching	Uniform			LB			LB-Perf		
	Avg	StdDev	T_{stat}	Avg	StdDev	T_{stat}	Avg	StdDev	T_{stat}
LRU	0.98	1.53	20h	0.78	0.77	3h	0.71	1	10h
p-LCP	0.43	0.84	>55h	<0.4	0.6	>55h	0.34	0.48	55h
LCD	0.5	1	4h	0.48	0.82	3h	0.4	0.7	3h
LAC	0.47	0.92	>55h	<0.4	0.65	>55h	0.36	0.6	>55h
LAC+	0.51	0.93	12h	0.47	0.65	9h	0.33	0.43	9h
									FOCAL

Fig. 3. Linear Topology with forwarding branches: Steady state values.



(a) Uniform. (b) Load Balanced (LB). (c) LB with Persistent Forwarding (LB-Perf).

Fig. 4. Linear Topology with forwarding branches: (a)-(c) Mean delivery time w.r.t content rank.

be observed under LB-Perf w.r.t. uniform forwarding and particularly for FOCAL (LAC+ with *LB-Perf*) as shown in Fig.4(c). Overall, the values reported in Tab.3 allow to quantify in 92% the gains brought by caching alone (LAC+) in terms of average delivery time reduction and in another 40% reduction via *LB-Perf*. FOCAL also achieves the lowest variance with good convergence time when compared to other approaches (absolute values of convergence are due to the slow request rate considered in simulations).

B. Fat tree with direct access to content repositories

We include this scenario to study FOCAL behavior in hierarchical networks with several paths with or without in-path caching opportunities. We consider the fat tree topology in Fig.2(b), with caches in every node storing 40 objects each. Content requests follow a Poisson process with intensity $\lambda = 1$ object/s. They uniformly address two repositories (Node 13 and 14 in Fig.2(b)), each one hosting a distinct catalog of $20,000$ objects, under the prefixes */Orange/* and */YouTube/*, ranked according to the same Zipf distribution. Two popularity profiles are considered in independent simulation runs: Zipf's skewness $\alpha_1 = 0.9$ and $\alpha_2 = 1.1$. To appreciate the impact on forwarding/caching, under Zipf's α_1 , the 400 most popular content items account for 50% of the traffic. This number drops to 30 with α_2. The presence of 10Mbps direct links to the repositories enriches the set of available paths, by making caching opportunistic: a node may choose not to forward Interests through the network of caches, rather to use the auxiliary direct link. Such configuration permits to understand whether and for which part of the catalog, in-network caching can be important to reduce end-user latency.

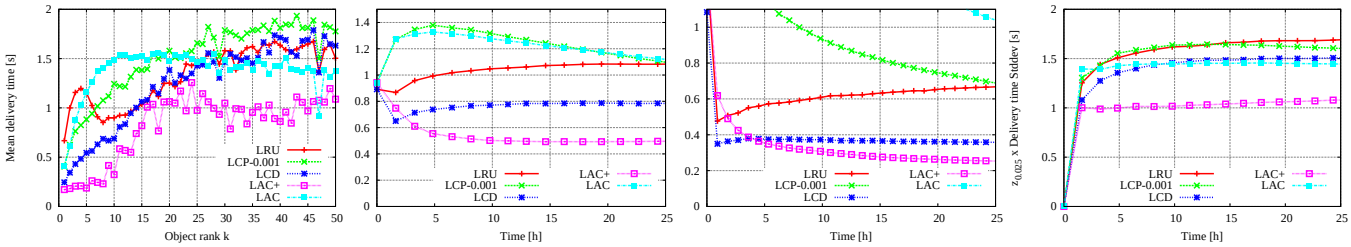
We report performance measures for this scenario in Fig.5. In this setting, FOCAL proves to outperform all other mechanisms under different metrics: it provides the best delivery time in average and standard deviation, attained within few hours. Such time scale is a typical busy period in access networks where caching performance would be mostly solicited in practice.

FOCAL is also robust to different workloads (here represented by two α factors) in contrast to other mechanisms like ϵ-LCP that fail to provide an acceptable performance bound. Indeed, when $\alpha = 1.1$ the average delivery time of ϵ-LCP converges two times slower to an almost two times higher value than what achieved by FOCAL. When $\alpha = 0.9$, ϵ-LCP provides more than two times higher latency than FOCAL, with very poor convergence time. On the other hand, looking at the way content items are managed by the different mechanisms, we observe a significant performance improvement for highly popular items, attaining gaps of a factor of five when $\alpha = 0.9$ as reported in Fig.5(a).

C. US backbone-like scenario

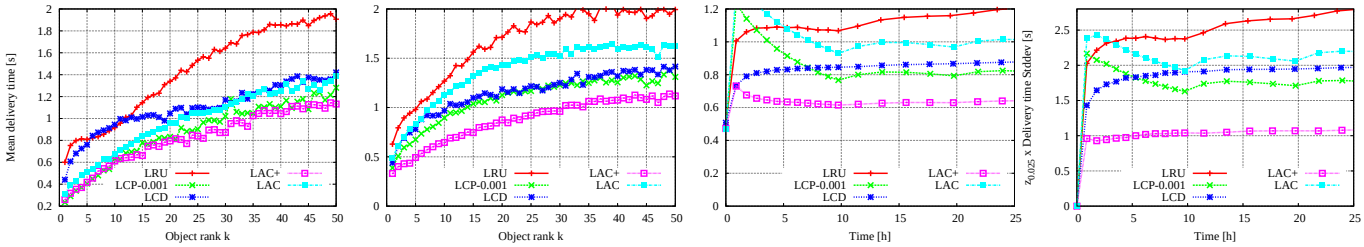
A backbone-like topology is made of core nodes which are access gateways to all clients attached to it (we build upon Abilene topology). Routing is much less hierarchical when compared to previous scenarios and Interest/Data traffic can flow in any direction. In such setup, node cache size is equal to 40 content objects. Links in the access are set to 500Mbps, and from 15Mbps to 100Mbps in the core. In this scenario we use three content repositories, each containing $20,000$ objects. We give objects at Repository 4 the prefix */Netflix/*, Repository 8 the prefix */Orange/* and Repository 10 */YouTube/*. Clients are equally interested in every catalog, i.e. every client addresses every catalog with probability $1/3$. Client requests follow a Poisson process with intensity $\lambda = 2$ objects/s. Each repository (also referred to as producer) is queried by clients following a Zipf-distributed workload with skewness equal to 0.9. Fig.2(c) summarizes the details of the network setup. While the content ranks have been so far fixed for the whole simulation, in this set of simulations, we confront the algorithms to a non-stationary content popularity distribution to introduce time locality. This is obtained by shuffling popularity rank every ten hours for every object in a given catalog. Every content's popularity rank changes over time while ranks remain Zipf-distributed. Note that this is far from being unrealistic. It widely pertains to real-world traffic where per-time-slot content popularity prevails [5].

FOCAL clearly outperforms other algorithms with or without temporal locality in the request workload. From Fig.6(b) to 6(d), delivery time performance results demonstrate it improves LCD and $\epsilon-$LCP ($\epsilon = 10^{-3}$) almost as much as they improved the basic LRU policy. More striking, FOCAL reduces LB-Perf + LRU average delivery time by 50% and stabilizes the delivery time in reducing its standard deviation



(a) Mean vs Rank,α_1. (b) Mean vs Time, α_1. (c) Mean vs Time, α_2. (d) Stddev vs Time, α_1.

Fig. 5. Fat tree topology, $\alpha_1 = 0.9, \alpha_2 = 1.1$



(a) Mean vs Rank, stationary. (b) Mean vs Rank, non stationary. (c) Mean vs Time, non stationary. (d) Stddev vs Time, non stationary.

Fig. 6. Abilene-like topology with stationary (s) and non-stationary (ns) workload.

by 60%. By comparing Fig.6(a) to Fig.6(b) we observe that FOCAL catches temporal locality very well. This is due to the fact that the mechanism adapts very fast to new conditions both in terms of popularity and network congestion.

V. Conclusion and Future Work

The paper explores ICN techniques for end-user delay minimization via latency-aware forwarding and caching strategies. Based on the insights on latency-aware caching alone and on the interplay with load balancing forwarding, we introduce FOCAL, an approach combining novel caching and forwarding strategies to jointly reduce end-user experienced latency with no network signaling nor coordination between routers. FOCAL combines a latency-proportional probabilistic caching policy, with a load-aware dynamic forwarding strategy, that preferentially routes popular content requests through a single path (set of caches), while globally achieving minimum load, thus delay minimization. By means of ICN simulations, we show that our proposal may achieve significant latency reduction (e.g. up to 60% average/variance delay reduction over LRU with LB-Perf in Fat Tree scenario), coupled with faster convergence w.r.t. solutions based on probabilistic caching approaches. In presence on non-stationary phenomena, FOCAL outperforms all other approaches demonstrating high self-adaptiveness to varying traffic/network conditions.

Such promising results encourage us to thoroughly characterize its dynamics by means of analytical models in a future work. Also, we plan to experiment with FOCAL in a mobile network setting where mobility may impose additional constraints on latency minimization.

References

[1] M. Badov, A. Seetharam, J. Kurose, V. Firoiu, and S. Nanda. Congestion-aware caching and search in information-centric networks. In *Proc. of ACM ICN*, 2014.

[2] G. Carofiglio, M. Gallo, L. Muscariello, M. Papalini, and S. Wang. Optimal Multipath Congestion Control and Request Forwarding in Information-Centric Networks. In *Proc. of IEEE ICNP*, 2013.

[3] G. Carofiglio, L. Mekinda, and L. Muscariello. Lac: Introducing latency-aware caching in information-centric networks. In *Proc. of IEEE LCN*, Oct. 2015.

[4] S. Eum, K. Nakauchi, M. Murata, Y. Shoji, and N. Nishinaga. CATT: Potential Based Routing with Content Caching for ICN. In *Proc. of ACM SIGCOM ICN Workshop*, 2012.

[5] C. Imbrenda, L. Muscariello, and D. Rossi. Analyzing Cacheable Traffic in ISP Access Networks for Micro CDN Applications via Content-centric Networking. In *Proc. of ACM ICN*, 2014.

[6] A. Ioannou and S. Weber. Towards on-path caching alternatives in information-centric networks. In *Proc. of IEEE LCN (Poster)*, 2014.

[7] N. Laoutaris, H. Che, and I. Stavrakakis. The LCD interconnection of LRU caches and its analysis. *Elsevier Science, Performance Evaluation*, 2006.

[8] Z. Li and G. Simon. Cooperative caching in a content centric network for video stream delivery. *Journal of Network and Systems Management*, 23(3):445–473, 2015.

[9] V. Martina, M. Garetto, and E. Leonardi. A unified approach to the performance analysis of caching systems. *CoRR*, abs/1307.6702, 2013.

[10] Z. Ming, M. Xu, and D. Wang. Age-based cooperative caching in information-centric networks. In *Proc. of IEEE INFOCOM NOMEN Workshop*, 2012.

[11] I. Psaras, W. K. Chai, and G. Pavlou. Probabilistic in-network caching for information-centric networks. In *Proc. of ACM SIGCOMM ICN Workshop*, 2012.

[12] V. Sourlas, P. Flegkas, and L. Tassiulas. A novel cache aware routing scheme for information-centric networks. *Elsevier Science, Computer Networks*, 59:44–61, Feb 2014.

[13] Y. Wang, Z. Li, G. Tyson, S. Uhlig, and G. Xie. Optimal cache allocation for content-centric networking. In *Proc. of IEEE ICNP*, 2013.

[14] E. Yeh, T. Ho, Y. Cui, M. Burd, R. Liu, and D. Leong. VIP: A Framework for Joint Dynamic Forwarding and Caching in Named Data Networks. In *Proc. of ACM ICN*, pages 117–126, 2014.

[15] Y.-T. Yu, F. Bronzino, R. Fan, C. Westphal, and M. Gerla. Congestion-aware edge caching for adaptive video streaming in information-centric networks. In *Proc. of IEEE CCNC Conference*, Jan 2015.