Multi-Terabyte and Multi-Gbps Information Centric Routers

Multi-Terabyte and Multi-Gbps Information Centric Routers

G. Rossini[1], D. Rossi[1], M. Garetto[2], E. Leonardi[3]
[1]Telecom ParisTech/LINCS, France – {giuseppe.rossini,dario.rossi}@enst.fr
[2]Università di Torino, Italy – michele.garetto@unito.it
[3]Politecnico di Torino, Italy – emilio.leonardi@polito.it

***Abstract*—One of the main research directions along which the future Internet is evolving can be identified in the paradigmatic shift from a network of hosts toward a network of caches. Yet, several questions remain concerning the scalability of individual algorithms (e.g., name based lookup and routing) and components (e.g., caches) of these novel Information Centric Networking (ICN) architectures. Exploiting a peculiar characteristics of ICN (i.e., the fact that contents are split in chunks), and the nature of video streaming (which dominates Internet traffic), this paper proposes a novel two-layers caching scheme that allows multi-Terabyte caches to sustain content streaming at multi-Gbps speed. We model the system as an extension, to the case of chunked contents, of the well known Che approximation, that has the advantage of being very simple and accurate at the same time. Simulations under synthetic and realistic trace-driven traffic confirm the accuracy of the analysis and the feasibility of the proposed architecture.**

I. Introduction

Recently, a healthy and hearty scientific discussion has sprout around the feasibility and interest of Information Centric Networks (ICN). Shortly, ICN architectures propose to recenter network design around *named content*, rather than the *addressable entities* of the current TCP/IP inter-networking paradigm. To allow efficient dissemination, content is split in *chunks*, transparently (and possibly pervasively) cached at intermediate routers along the path that leads user requests to a permanent copy of the content of interest.

Fervents of ICN have adopted this new paradigm with sometimes unjustified enthusiasm. Opponents of ICN have shown an equally strong pugnacity. On the one hand, the benefits of ICN may be exaggerated, in that most performance evaluation studies consider rather small catalogs, exiguous caches and large popularity skews [1], [2]. On the other hand, the limits of ICN may be exaggerated as well, in that they mostly rely on the argument that the cache to catalog size ratio is extremely unfavorable, given technological limits of the former and the restless growth of digital data [3].

In practice, the success of ICN architectures depends on their ability to provide (i) large caches, able to (ii) process data traffic at line speed. It has been pointed out that these two requirements tradeoff: due to technological limits of currently available memory technologies, state-of-the-art sizes to about 10GB [4], [5] the largest cache size that can sustain over 10Gbps speeds. Our work breaks this limit by proposing a novel scheme for the management of two-layer caches that exploits a peculiarity of ICN. Taking advantage of correlation among chunks, we use request arrivals in the ICN data plane as predictors of requests for subsequent chunks. We then proactively move from a large but slow memory (e.g., SSD) a number of chunks to a fast but small memory swap area (e.g., DRAM). Additionally, by batching memory transfer operations over multiple chunks, we transition from an operational point where SSD is dominated by memory access time, to an operational point where SSD external data rate is the bottleneck – gaining over an order of magnitude in terms of sustainable data rates. Finally, we optimize the system to yield low startup latency (e.g., for video streaming) by storing the first chunk of a large portion of the catalog directly in fast memory (trading with swap area).

Summarizing our contributions, we (i) propose a simple yet effective hierarchical scheme that scales up ICN caches to multi-Terabyte size while maintaining multi-Gbps line-speed operation, (ii) model the system with a simple yet instructive extension of the Che approximation for chunked content, and (iii) perform a thorough evaluation of system performance, comparing analytical results to trace-driven and synthetic simulation results.

II. Background

The aim of this section is not only to overview related work (Sec. II-A), but also to present a realistic operational point we can expect our system to work at. Given the tremendous importance of video – see the well known Cisco Visual Networking Index for often cited figures and trends – it makes sense to consider Video-on-Demand (VoD) as the main driver for ICN, or at least as a very popular service ICN will have to deal with. While nowadays Internet video is mostly low quality, we point out (Sec. II-B) that any technological shift, such as the one toward ICN, must be future-proof with respect to the likely video quality evolution (and hence bitrate increase). We furthermore argue (Sec. II-C) that ICN devices that are built today will have to last for several years, so that our design conservatively employs memory technologies that are currently on the market.

A. State of the art

Much work on ICN has focused on algorithms and protocols for a network of caches. Among the different ICN proposals overviewed in [6], CCN is the one that has gained more popularity in the scientific community [7]. While our work has a broader scope than CCN, however due to CCN popularity and to the lack of a commonly agreed ICN terminology, in this work we adopt the CCN jargon, that we briefly introduce.

CCN clients request Data in a pull-based fashion sending *Interests* for named contents. Requests are *forwarded* hop-by-hop toward a permanent copy of the requested Data: for each Interest, CCN nodes perform lookup for content names in a *Forwarding Information Base (FIB)*, that stores the set of interfaces through which any given content can be reached. As multiple paths are possibly stored for any given name, a *Strategy Layer* is responsible for the selection of one (or more) next-hop interface among the set of possible ones. CCN nodes along this path may possess cached copies of the content of interest within their own *Content Store*: in this case, Interests do not need to reach the permanent copy stored at the repository, and the temporary copy in cache is directly sent back to the client along the reverse path. Indeed, Data travels back toward the requester following a trail of bread-crumbs, that are stored in a *Pending Interest Table (PIT)* at every network node.

While most ICN work focuses on algorithmic, protocol or performance aspects, fewer studies exist that address system design [4], [5], [8]–[12], and that are thus closer in spirit to our work. Specifically, seminal papers [4], [5] tackle the design of an ICN router by considering FIB, PIT and caching scalability, with more recent work refining aspects of FIB [5], PIT management [9]–[11] and name lookup [12]. As work on caches [4], [5] is closer to ours, we will provide a more detailed comparison later on.

Other works on caching systems such as Fawn [13] and HashCache [14] are worth citing, since we leverage memory indexing techniques employed there, similarly to what was done in [5]. At the same time, the operational point considered in Fawn and HashCache is rather far from high-performance ICN, so a direct comparison does not apply. Conversely, we emphasize that work on hierarchical caches [15], [16], is not quite related to our proposal, as "hierarchy" relates there to the topological position in a cache "network". Furthermore, while hierarchical memory systems have been long studied in the context of computer architectures [17], the typical workload in that context (e.g., reading inputs, dot product, shuffle exchange, merging two sorted lists, etc.) is rather different from the sequence of interests generated by video traffic portals that we consider here. Hence, we point out that none of previous works proposes a specialized two-layer architecture like ours, that increases by several orders of magnitude the storage capacity of router caches, leading to tangible advantages for ICN architectures.

B. Video

Nowadays Internet streaming is dominated by portals such as YouTube, Kankan, Hulu, etc. In terms of networking, video traffic can be specified by the (average) streaming rate ρ that the network needs to sustain in order to avoid stutter in the playback. The streaming rate ρ depends, on the one hand, on technological limitations of the physical display, and, on the other hand, on the availability of content encoded at that resolution.

Despite physical resolution steadily increases (e.g., 4K displays were recently shown at CES2013), such extremely high-definition content is not readily available in the consumer market. As such, most of the freely available Internet content is generally streamed at a much lower rate. For instance, the default video resolution of popular free services such as YouTube is 640×360, is encoded with H264 at a median rate of 500Kbps, according to measurements in [18]. Even popular paying services such as Netflix [19], offer only a minor part of their content at HD (1280×720, 5Mbps) or FHD quality (1920×1080, 7Mbps). Additionally, advances in video codecs will likely reduce the streaming rate required for the same perceptual video quality (e.g., H265/HEVC encoding of FHD videos requires 3Mbps, or less than half than H264 at the same perceptual quality [20]). Overall, we argue that an explosion of bandwidth requirement is unlikely owning to the fact that changes in screen sizes happen very slowly [21], and we can expect the resolution growth to be compensated by advances in encoding technologies.

Based on the above observations, we consider two operational points: a current one with $\rho = 500$Kbps video streaming rate, and a future one with $\rho = 5$Mbps. Additionally, given increasing user adoption, we consider advanced viewing functionalities, such as the ability to pause, rewind or fast-forward the video, as essential features of a VoD service – hence, these functionalities are explicitly accounted for in our system design and evaluation.

TABLE I
MEMORY TECHNOLOGIES CHARACTERISTICS

	HDD [22]	SSD [23], [24]	DRAM [5]	SRAM [5]
Ext data rate (Gbps)	1.4	20-24 (64-128KB)	- -	- -
I/O data rate (Gbps)	6	64	-	-
I/O interface	SATA	PCIe	-	-
Max Capacity (GB)	4000	2000	10	0.026
Avg latency (μs)	4000	30-78 (4KB)	0.05	0.00045 -
Speed (rpm)	7200	-	-	-
Block size (KB)	4-32	4-512	-	-
Allocation unit	Sector	Page	-	-
Allocation size	512B	4KB	-	-
Price (USD/MB) [5]	0.00005	0.003	0.02	27

C. Storage

Opponents of ICN pinpoint the disproportion between the raw volume of digital data consumed worldwide against the size of memory technologies available for real-time ICN operations. Their critic is however hitting the right spot, as ICN success is conditioned to the feasibility of large content stores. As such, it would be desirable to include technologies such as SSD (or HDD) to scale up the cache size up to some TB (see Tab. I).

However, as pointed out in [4], [5], technological constraints nowadays imply a tradeoff between the size of the storage and its access speed. Both [4], [5] identify the single-layer system comprising a fast-but-small SRAM index addressing a large-but-slower DRAM cache storage, as the largest and fastest architecture for today's ICN caches, which practically limits to about 10GB the size of caches that can operate above 10Gbps.

Notably, two main performance aspects need to be considered for ICN caches. First (i), ICN routers perform lookup on the content names, in order to decide whether an interest can be satisfied locally by cached content, or whether it needs to be forwarded. In case of a match, (ii) the lookup returns a pointer

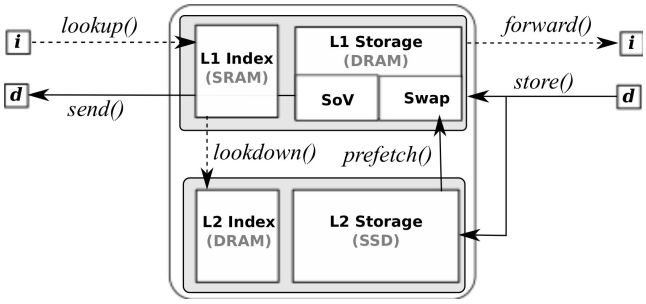

Fig. 1. Synopsis of 2-layer ICN caching architecture.

to the memory region storing the content itself, that needs to be transferred to assemble a response packet.

Operation (i) requires to move small amount of data, and its main performance indicator is thus the *memory access latency*. In order to sustain line rate operations, the memory index must be accessed at a frequency that in the worst case matches the throughput of the maximum sustainable data rate. In other words, assume that the aggregate router data rate is R, that data chunks have size S_c, then the maximum number of chunks that can be transferred in the time unit is $\lceil R/S_c \rceil$, that also upperbounds the number of accesses to the index, so that $t_{access} \leq 1/\lceil R/S_c \rceil$. Operation (ii), instead, depends on the chunk size, and can be either driven by access latency (in case of small chunks) or by the *external data rate*, i.e., the sustainable throughput when reading random memory blocks (in case of large chunks).

In [5], a core ICN router is designed to support 8x40Gbps interfaces serving S_C=1500B chunks, stored on a 10GB DRAM cache implementing sequential access and LRU replacement policy, indexed on a 100MB SRAM chip by employing H=40bits long content names hashes. Interestingly, [5] also proposes an edge router design where a 1TB SSD cache is indexed by a 10GB DRAM chip. As [5] employs small size chunks, in both cases operations (i) and (ii) are bottlenecked by memory access latency: for the edge router case, [5] reports a maximum access rate of 0.417Mbps, corresponding to 5Gbps when S_C=1500B (this figure may vary as RamSan-70 [24] has 30μs latency, so it would only support 0.5Gbps).

Yet, from Tab. I we notice that a relatively larger memory read size (64KB-128KB) would allow to achieve high SSD memory throughput (20Gbps-24Gbps). HDD technology is instead not so appealing due to limits on both the achievable data rate (1.4Gbps), and the maximum data rate of the SATA interconnection (6Gpbs), so we disregard it in the following. Hence, we argue that a system design *moving SSD bottleneck from memory access time to external transfer rate* could possibly gain over an order of magnitude in terms of the sustainable rate (e.g., from 4x-5x considering the unspecified memory in [5] to 40x-48x considering the RamSan-70). Additionally, given large SSD size, we expect favorable performance in terms of caching as well.

III. System Architecture

Especially, [5] implicitly states an important paradox in ICN that raises due to the tradeoff between memory size and speed: while in principle as the cache size grows so does the cache hit probability, however in practice the cache would need to be accessed more frequently, which is only possible for small caches. We argue that the above limits can be worked around by (i) exploiting temporal correlation in the interest request stream to compensate for memory access latency, by (ii) batching operations on consecutive chunks to ensure high-speed memory transfers and by (iii) introducing a multi-layer hierarchical caching system exploiting the intrinsic differences of heterogeneous memory technologies (Tab. I).

We overview the system design with the help of Fig. 1. At high level, the system is organized as a hierarchy of caches: a (multi-Terabyte) L2 cache masked behind a small L1 cache able to operate at (multi-Gbps) line rate. The idea is that L1 should contain chunks of ongoing videos for fast service in the data plane. Conversely, L2 is large but slow, so it can store a significant portion of the catalog, that it needs however to opportunely prefetch to L1.

In ICN, objects are split in multiple named chunks, so that a request for a named chunk can be used as predictor of subsequent requests for other chunks belonging to the same object. This suggests to proactively trigger prefetching operations, operating over batches of chunks, so as to sustain high transfer throughput at L2. At the upper level, as the content has been prefetched, operations on index and cache memory happen at line speed. At the lower level, cache operations happen before the request for the next batch comes. This effectively decouples the timescale of the lower, slower, level from the timescale of the network data path on the upper level.

At each level, caches are organized into a classical (index,storage) pair: at L1 a SRAM chip indexes a DRAM cache, while at L2 a DRAM chip indexes an SSD cache. Cache *indexes* are accessed at each name lookup in the data-plane: whenever an interest packet hits the router, if the name is present in the index, the index contains the memory location of the corresponding named data in the *storage* memory. While it is out of the scope of this paper to select an indexing technique, that could be any among the hash-based exact-match lookup techniques proposed in [14], with hash sizes opportunely tuned as in [4], [5].

To further optimize for ICN specificities, we divide L1 storage in two areas: (i) *Start of Video* (SoV), that is rather static and keeps the first chunk of popular videos, and (ii) a *Swap* area, that is rather dynamic and prefetches from L2 chunks of ongoing downloads. Without loss of generality, we consider that L1 SoV, L1 Swap and L2 caches are managed by a Least Recently Used (LRU) replacement policy.

A. System design

Let us denote with c_i the i-th chunk of object m in a catalog $\mathcal{M}$ comprising M movies. We consider all chunks to have equal size S_c (possibly using padding for, e.g., the last chunk of an object). Requests for the first chunk c_1 of an object $m \in \mathcal{M}$ can be assumed to be independent of previous chunk requests, whereas requests for subsequent chunks c_i ($i > 1$) of the same object are highly correlated in time: in the case of video, we can loosely assume chunks to be separated by a gap $\delta t = S_c/\rho$ related to the streaming rate (we will deal with the chunk arrival process in Sec. IV).

Let us consider the first chunk c_1 of a new object. Whenever an interest for c_1 hits the ICN router, a *lookup()* is issued to the L1 cache first: in case of a hit in the SoV cache (i) a frame containing c_1 data is assembled and returned via *send()*; (ii) the system signals L2 to *prefetch()* a batch of B consecutive chunks to the L1 Swap (and the L1 index is updated accordingly). Notice that especially critical to our design is the ability to move video chunks from L2 to the L1 swap and update the L1 index before the corresponding requests effectively hits the cache.

In case of a L1 miss for c_1, content search continues both (i) in the L2 cache within the ICN router via a *lookdown()* and/or (ii) in the router vicinity via interest *forward()*. Operations (i)-(ii) can be performed in parallel (to optimize user delay) or in sequence (to save network traffic). Sequential operations trigger network traffic only in case of a L2 miss, but add in this case a delay proportional to the *lookdown()* time in the L2 index. Parallel operations avoid this delay but possibly generate unnecessary network traffic in the case of a L2 hit. We argue that parallel operations are preferable as they lead to better user experience, and since the overhead is limited to the first chunk of an object (in our settings, 10KB of a 10MB video corresponds to a 0.1% overhead). Moreover, parallel operations can also simplify system implementation. Casting our design to CCN, in case of a miss, the node adds an item to the PIT and forwards the request. Internally, the L2 cache can then be simply seen as another CCN "face" along which to forward the request (i.e., *lookdown()*), and in case of a hit, the corresponding data will satisfy the PIT request. This again decouples operation of L1 and L2, ensuring that data-plane forwarding operations can be performed at line rate.

Pushing the decoupling of L1 and L2 further, we notice that in principle it would be more efficient to let L1 SoV and L2 store completely disjoint chunk sets. Otherwise stated, from caching purposes, it would be useless to store a redundant copy of the first chunk in both L1 and L2, since the latter will never be used. However we also note that L2 cache is much larger than L1, and especially persistent: as such, this slight redundancy allows the system to easily recover after a failure, and promptly re-initialize the L1 SoV by transferring the first chunk of the most popular objects.

After having consumed the first chunk, the user will start requesting subsequent chunks. Due to prefetching, chunks $c_2 \dots c_B$ are now to be found in L1, so that whenever a request for chunk c_{B-1} arrives, the system can proactively lookdown for $c_{B+1} \dots c_{2B}$. In CCN terms, since the last portion of a chunk name represents the chunk sequence number, this is easy to do since upon reception of chunk c_k the *lookdown()* decision requires checking a simple modulus division $k\%B = B - 1$.

B. System optimization

This behavior holds as long as the user does not jump ahead in time, breaking the sequential view mode. While we expect sequential behavior to be vastly predominant, still we recognize that users increasingly depend on added value-services such as advanced VCR functionalities allowing them to pause, rewind or fast-forward, possibly generating non-sequential chunk request patterns. We therefore suggest additional system optimization that, while non-necessary for the correct operation, can provide benefits by pushing a moderate complexity to the edge of the ICN network.

Pause. In case the pause time is long enough (i.e., larger than L1 Swap characteristic time, see Sec. IV-B), content will disappear from L1 Swap. Thus, when the user restarts, typically the first interest packet sent by the user for the next chunk not yet on the user buffer will be propagated upstream. To increase the efficiency of the system, prefetching should extend to the user cache. When a user pauses a video, the decoder automatically goes on retrieving a few extra chunks before suspending the download. Such extra chunks are stored locally in the playout buffer of the user, and thus they are already available when the user wants to watch beyond the point of the suspension. As soon as playback resumes, the player should proactively send an interest for the next chunk not yet on the user buffer: this allows the system to restart the prefetching mechanism, transferring chunks from the slow cache to the swap area in the usual way. As a side note, as the content is very likely stored on L2, the player can avoid an unnecessary lookup to propagate beyond the first ICN router by limiting the scope (e.g., setting a low TTL) for the first interest after a pause (and only in case the scoped interest has failed to hit content in L2, after a time-out the user will send a new interest packet with unlimited TTL).

Fast-forward. In case of forward jumps larger than $B \cdot \delta t$, the request for the new chunk will generate a L1 cache miss, and operations occur as previously explained for the first chunk c_1. Yet, as the playback is not paused, nothing can prevent a small buffering delay in this case.

Rewind. This can be completely masked by the user cache: as previous chunks have already been played, they are still available in the user device, so no specific action is needed.

C. System dimensioning

Our system relies on a number of parameters that need to be set, notably the chunk and batch sizes, as well as the partition between SoV and Swap areas in L1 cache. Additionally, the feasibility of the operations outlined in the previous section, in terms of access latency and speed in the L2/L1 swap operations, needs to be checked.

Chunk and batch size. The chunk size is the minimum data object granularity. Small chunk sizes are preferable (e.g., to avoid paying a padding penalty for a myriad of small objects), though not necessarily so small to be fit in current MTUs (as initially proposed by CCN [7]). Clearly, as the chunk size determines the frequency of all system operations, a larger chunk size is also beneficial as it relaxes the system complexity. Currently ongoing effort [25] already employs larger chunk sizes of about 4KB, so that we consider chunk sizes of $S_c = 10\,\text{KB}$ to be a reasonable compromise. Yet, efficient memory transfers require larger read/write size, so that it is necessary to batch B chunks in order to achieve a batch size that allows sustaining high external data rates. As SSD achieves 20-24Gbps in random read of 64-128KB from Tab. I, it is safe to set $B = 10$ chunks.

SoV vs. Swap size. Observe that: (i) SoV relates to properties of the catalog, whereas Swap size reflects the maximum number of active flows; (ii) Swap must be sufficiently large to guarantee that prefetched chunks are not evicted before a time lapse equal to the watching time of a batch (i.e., 1.6s when $\rho = 500$Kbps and 160ms when $\rho = 5$Mbps).

Let us consider for simplicity an equal partitioning of L1 SoV and Swap. With 5GB worth of cache space, the SoV could keep the first chunk for 500,000 videos[1], while 5GB worth of L1 Swap space allows for 50,000 concurrent flows when BS_c=100KB, thus, potentially sustaining up to 20-24Gbps of aggregate streaming. Notice that these figures refer to the case in which all remaining chunks requested by users are retrieved from the L2 cache, and thus temporarily placed in the Swap. This is, indeed, the worst case for the Swap sizing.

We emphasize that, considering an average video size of 10MB, about 5TB would be needed to keep the top 500,000 videos of the nowadays YouTube catalog.

Since we expect caching performance to be mainly driven by L2 size, when routers operate at a data rate slower than 20-24 Gbps, it also makes sense to consider a smaller L1 DRAM cache, as this translates into cost reduction (mainly as L1 SRAM index shrinks) that could compensate, if not entirely cover, the additional SSD cost. For example, 1 GB of Swap would correspond to the minimum size needed to sustain a data rate of 4-4.5 Gbps. A 1 GB SoV, instead, would be able to store initial chunks for 100,000 top videos, responsible for over 58% of the requests.

L2 prefetching throughput. Our system is designed and engineered so that the L2 external data rate is the bottleneck: in practice, the internal L2 to L1 prefetching rate limits the aggregate data rate that can be transparently sustained by the hierarchical system. Indeed, since video chunks need to be prefetched from L2 to L1, the system can reasonably sustain about 20-24Gbps worth of aggregate streaming rate, corresponding to about 40,000-48,000 concurrent flows streamed at 500Kbps (coherent with the L1 Swap size). While the sustainable aggregate rate is already larger with respect to a mono-layer DRAM+SSD configuration [5], we also notice that the PCIe bus supports IO data rates up to 64Gbps, so that it may be worth investigating whether the use of multiple *parallel* SSD could translate into an additional gain.

L2 access latency. As L2 operations occur at a lower rate due to prefetching, these face less strict performance constraints. Roughly speaking, the batching factor $B = 10$ decimates the number of operations per second at L2 with respect to L1. The reception of a batch of chunks from the network triggers a *store()* operation (that also implies a DRAM index update) (as *store()* operation takes 50ns access latency, it does not constitute a bottleneck). To group chunks belonging to the same batch, individual chunks received from the network are first temporarily collected in the L1 Swap. Upon reception of the last chunk of a batch, the batch is atomically copied from L1 Swap to L2. Note that this does not invalidate previous Swap dimensioning considerations: indeed, by construction, the aggregate rate (i.e., from the network plus L2) at which chunks are written into the Swap does not exceed the line-speed. The reception of interests from the network triggers, in case of a L1 swap miss, a *prefetch()* operation for B chunks. At an external rate of 20Gbps, the first prefetched chunks would be transferred to L1 Swap after 4μs, i.e., significantly in advance with respect to the expected arrival time of the corresponding interest at the ICN node. Indeed, recall that inter-chunk arrival time $\delta t = S_c/\rho$ is about 160ms (16ms) in case of 500Kbps (5Mbps) videos.

L2 lifetime. While [4], [5] do not mention any drawback of SSDs, it has to be pointed out that as NAND dies shrink close to physical limits, this creates endurance and reliability issues: the lifetime of NAND based on SLC technology is estimated to 100K write cycles, fact that can be problematic for a caching architecture. Yet we notice that L2 to L1 transfers mostly read data from L2, that is written on L2 on a slower basis: as writing happens in random positions, it will take long before the SSD worns out (we estimate this time in Sec. V-C). Alternatively, algorithmic techniques to limit the data coming into L2 are easily envisionable (e.g., on the first miss, cache the content name on a DRAM; then, on a content name hit, cache the content itself on the SSD). Finally newer technologies such as PCM (phase-change memory) and ReRAM (resistive RAM), are expected to overcome those limits [26].

IV. System Model

We introduce a simple analytical model that describes the performance of our 2-layer ICN router architecture, with special attention to the bandwidth required from the rest of the ICN network (i.e., from the upper nodes of the cache network). For simplicity, we consider an edge (leaf) router that directly serves user-generated requests. Yet, the model could be used to analyze the performance of a core router receiving an aggregate miss stream of requests forwarded by one or more children nodes – hence it could serve as basic building block for the performance analysis of the whole network.

A. Traffic and user model

As previously stated, we focus on video traffic and assume that users request videos from a large fixed catalog $\mathcal{M}$ having size M videos, and that requests for each video $1 \leq m \leq M$ arrive according to a homogeneous Poisson process of rate λ_m. Let $\Lambda = \sum_m \lambda_m$ be the aggregate arrival rate of video requests.

Notice that our traffic assumptions are equivalent to describing the sequence of requests arriving at the cache by the well known Independent Reference Model (IRM), according to which object requests are i.i.d., following an assigned popularity law (usually Zipf). Without lack of generality we assume that videos are sorted in decreasing order of their request rate.

All VCR functionalities can be jointly modeled by specifying, for each chunk c of content m, a pair of probabilities ($p_{m,c}$,$q_{m,c}$). $p_{m,c}$ denotes the probability that chunk c will be eventually

[1]For a catalog size of M =500,000,000 videos with Zipf's exponent $\alpha = 1$, the SoV would store the initial chunk of top videos responsible for over 65% of the requests.

downloaded by the user as a result of its entire watching experience (i.e., including the impact of abandon and all jumps). Notice that $1 - p_{m,c}$ represents by construction the probability that chunk c is either skipped, or the content m download is aborted before chunk c is requested.

Instead, $q_{m,c}$ denotes the probability that chunk c is requested as a result of a jump-forward operation to a totally new point, such that chunks immediately preceding c have not been downloaded and locally cached by the user. In such a case we say that chunk c is delivered out of sequence. Notice that not all forward jumps necessarily contribute to $q_{m,c}$, but only those ending at a portion of video not yet downloaded. These jumps are especially critical because we assume that no prefetching is possible in this case before restarting the playback. In our performance analysis we will assume for simplicity that the above probabilities $p_{m,c}$ and $q_{m,c}$ are given[2].

B. Preliminary: the Che approximation

We briefly recall a simple yet powerful approach [15] to analyze the behavior of an LRU cache under the IRM, upon which our model builds on. Consider an LRU cache capable of storing C objects. Let $T_C(m)$ be the characteristic time needed for C distinct objects (not including m) to be requested by users. Therefore, $T_C(m)$ represents the *cache eviction time* for content m, i.e., the time since the last request after which object m will be evicted from the cache.

Che's approximation [15] assumes $T_C(m)$ to be a constant independent of the selected content m. This assumption has been given a theoretical justification in [27], where it is shown that, under a Zipf-like popularity distribution, the coefficient of variation of the random variable representing $T_C(m)$ tends to vanish as the cache size grows. Furthermore, the dependence of the eviction time on m becomes negligible when the catalog size is sufficiently large. Che's approximation greatly simplifies the analysis of caching systems because it decouples the dynamics of different contents: interaction among contents is summarized by T_C, which acts as a single primitive quantity representing the response of the cache to each individual object request.

More in detail, Che's approximation states that an object m is in the cache at time t, if and only if a time smaller than T_C has elapsed since the last request for object m, i.e., if at least a request for m has arrived in the interval $(t - T_c, t]$. Under the assumption that requests for object m arrive according to a Poisson process of rate λ_m, the time-average probability $p_{\text{in}}(m)$ to find object m in the cache is given by:

$$p_{\text{in}}(m) = 1 - e^{-\lambda_m T_c} \tag{1}$$

As immediate consequence of PASTA property for Poisson arrivals, observe that $p_{\text{in}}(m)$ represents, by construction, also the hit probability $p_{\text{hit}}(m)$, i.e., the probability that a request for object m finds object m in the cache. Considering a cache of size C, by construction:

$$C = \sum_{m,c} \mathbb{I}_{\{m \text{ in cache}\}}$$

[2] $p_{m,c}$ and $q_{m,c}$ could be computed by developing a detailed microscopic model to account for user behavior, which is out of the scope of this paper.

After averaging both sides, we obtain:

$$C = \sum_{m} \mathbb{E}[\mathbb{I}_{\{m \text{ in cache}\}}] = \sum_{m} p_{\text{in}}(m). \tag{2}$$

The only unknown quantity in the above equality is actually T_C, which can be obtained with arbitrary precision by a fixed point procedure. The average hit probability of the cache is then:

$$p_{\text{hit}} = \sum_{m} \frac{\lambda_m}{\Lambda} p_{\text{hit}}(m) \tag{3}$$

C. Analytical model of two-layer caching architecture

To analyze the performance of a router based on our proposed architecture, we extend and generalize the Che approximation recalled above. With respect to the existing model, which has been developed for a single LRU cache under IRM, we need to take into account: (i) the separation of video m into l_m chunks of size S_c; (ii) the impact of VCR functionalities; (iii) the presence of three different LRU caches: the SoV and Swap L1 DRAM caches, and the L2 SSD cache.

Let $p_{\text{hit}}^{\text{SoV}}(m,c)$, $p_{\text{hit}}^{\text{Swap}}(m,c)$ and $p_{\text{hit}}^{\text{L2}}(m,c)$ the hit probability of chunk (m,c) in the SoV, Swap and L2 cache, respectively. We denote by T_C^{SoV}, T_C^{Swap} and T_C^{L2} the cache eviction time of SoV, Swap and L2, respectively.

Observe that SoV receives, by construction, only requests for the initial chunk of each video, i.e., $\lambda_{m,c}^{\text{SoV}} = 0$ if $c > 1$ and $\lambda_{m,c}^{\text{SoV}} = \lambda_m$ if $c = 1$. On the contrary, the Swap and L2 memories receive only the requests for non-initial, actually downloaded chunks, hence we have $\lambda_{m,c}^{\text{L2}} = \lambda_{m,c}^{\text{Swap}} = \lambda_m p_{m,c}$ if $c > 1$, whereas $\lambda_{m,c}^{\text{L2}} = \lambda_{m,c}^{\text{Swap}} = 0$ if $c = 1$.

Now, the SoV cache can be modeled as a standard LRU cache storing only the first chunk of each video, for which we can apply exactly the model described in Sec. IV-B. Hence,

$$p_{\text{hit}}^{\text{SoV}}(m,1) = \frac{\lambda_m}{\Lambda}(1 - e^{-\lambda_m T_c^{\text{SoV}}}) \tag{4}$$

The Swap area and the L2 cache are also managed by LRU and they are organized in chunks, but to analyze their behavior we need to consider that (i) the arrival process of chunks does not follow the IRM model, as well as the impact of (ii) prefetching and (iii) VCR functionalities.

With respect to point (i) above, the request for the first chunk of video m triggers the subsequent arrival of other $l_m - 1$ requests, equally spaced by δt, for the other chunks belonging to the video: hence the aggregate sequence of requests is not i.i.d. Yet, we can observe that, independently from the detailed structure of the process according to which requests for the different chunks of a video arrive at the cache, as immediate consequence of the fact that videos are initially requested by users according to a Poisson process, also the request process of each individual chunk (m,c) turns out to form a Poisson process with rate $\lambda_{m,c} = \lambda_m\, p_{m,c}$. This key observation allows us to immediately extend the classical Che's approximation to the Swap and L2 caches:

$$p_{\text{hit}}^{\text{Swap}}(m,c) = \frac{\lambda_m p_{m,c}}{\Lambda'}(1 - e^{-\lambda_m p_{m,c} T_c^{\text{Swap}}}) \quad (c > 1) \tag{5}$$

$$p_{\text{hit}}^{\text{L2}}(m,c) = \frac{\lambda_m p_{m,c}}{\Lambda'}(1 - e^{-\lambda_m p_{m,c} T_c^{\text{L2}}}) \quad (c > 1) \qquad (6)$$

where $\Lambda' = \sum_m \sum_{c>1} \lambda_m p_{m,c}$.

Concerning prefetching (ii), we remark that the hit probability of the Swap area, as computed above, must be given a special interpretation. Indeed, in terms of hit probability the Swap behaves as if it received an arrival process of requests analogous to the one arriving at L2. However, this (virtual) process includes also the fictitious requests generated by the prefetching mechanism. Thus, actual requests for prefetched chunks do not actually generate any miss (i.e., chunks are deterministically found in the Swap area). Instead, $p_{\text{hit,swap}}(m,c)$ represents a real hit probability for those chunks for which prefetching is not possible (i.e., chunks that are requested out-of-sequence).

Finally, we incorporate VCR functions (iii) observing that: the request for the first chunk of any video m must be forwarded whenever it is not found in the SoV cache; every chunk $c > 1$ downloaded by the user, which is not present in L2, must be proactively retrieved by the router and stored into L2; every chunk $c > 1$, which is requested out-of-sequence (e.g., for effect of a jump-forward operation) must also be retrieved externally, except in the fortuitous case in which it is found in the Swap area (for effect of other users requesting it nearly at the same time). Notice that out-of-sequence requests must be forwarded even if the corresponding chunks are present in the L2 cache (as requests must be served in the data plane, interest *forward()* operation in the network is performed in parallel to *lookdown()* in L2).

Overall, we can compute the aggregate bandwidth requested by the router from the rest of the cache networks:

$$\begin{aligned} B_s = \sum_m \lambda_m S_c (1 - p_{\text{hit}}^{\text{SoV}}(m,1)) + \\ + \sum_{m,c>1} \lambda_{m,c}\, S_c\, q_{m,c} (1 - p_{\text{hit}}^{\text{Swap}}(m,c)) + \\ + \sum_{m,c>1} \lambda_{m,c}\, S_c\, (1 - q_{m,c})\, (1 - p_{\text{hit}}^{\text{L2}}(m,c)) \end{aligned} \qquad (7)$$

where the first term accounts for *forward()* operations due to misses in SoV, the second term is due to forward jumps and the third term, that we expect to be dominant over the formers, accounts for misses in L2 cache.

V. Performance evaluation

We now validate the model (Sec. V-A), and contrast its predictions against simulation results obtained under both synthetic (Sec. V-B) and realistic (trace driven) arrival patterns (Sec. V-C).

We carefully build a synthetic scenario as follows. The YouTube catalog was observed having about 100 million videos [28] (much larger than what considered in [29]), having average size 10 MB [30] and median rate of 500Kbps [18]. To gather conservative results, we consider catalogs of over $M = 10^8$ videos (up to $M = 5 \cdot 10^8$ to account for catalog growth). We also consider two video streaming rates: a current scenario with low video quality (500Kbps, 10MB average video size) and a future one with very high quality (5Mbps, 100MB average video size).

In terms of video popularity, it is worth noticing that values of Zipf's exponent α close to 1 have been recently shown [29] to provide best fitting with experimental data over daily time scales (which are the most relevant for caching)[3]. Yet, to gather more conservative results, we also consider less skewed Zipf's exponents $\alpha \in [0.8, 1]$.

In reason of our system dimensioning, we consider a L1 SoV+Swap cache in the range 2-10GB and a L2 cache in the range 1-10TB range, while we fix the chunk size to S_c=10KB, and batch operations in groups of B =10 chunks. We highlight that, as long as the Swap area size meets the minimum requirement early discussed and the popularity law is stationary, L2 cache performance are completely insensitive to the requests arrival rate (both in terms of hit probability and normalized bandwidth saving). For these reasons, in what follows we consider a normalized average request rate $\Lambda = 1$.

A. Model validation

We first validate a fundamental property of our model, namely, the insensitivity of analytical results with respect to the inter-chunk arrival time δt. This is indeed a key feature of our analysis, as it allows us to apply Che's approximation even if the chunk arrival process does not satisfy the IRM. We consider the current streaming scenario, with Zipf's law exponent $\alpha = \{0.8, 1.0\}$. For the time being, we do not account for VCR functions and set $p_{m,c} = 1$ for each chunk.

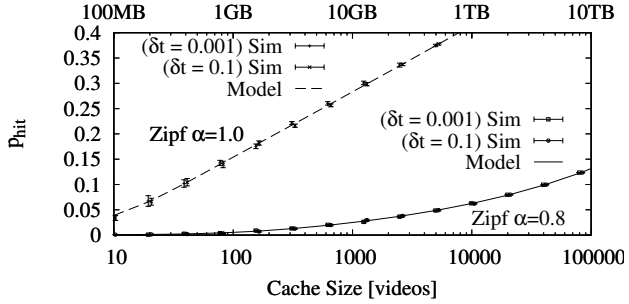

Fig. 2. Hit probabilities obtained by analysis and simulation for chunk inter-arrival time $\delta t = \{0.001, 0.1\}$, catalog size $M = 10^8$, Zipf $\alpha = \{0.8, 1\}$.

We then set δt equal to either 0.001 or 0.1, to consider two very different scenarios: the first one corresponds to the case in which all chunks are requested within an interval equal to the average video inter-request time (strong correlation in the chunk arrival process), while in the second scenario chunks belonging, on average, to 100 different flows arrive intermingled together[4] (weak correlation in the chunk arrival process). Results are reported in Fig. 2, in which analytical prediction is compared against 95%–level confidence intervals obtained from detailed event-driven simulations at chunk level. The excellent agreement

[3]Note that estimates of α over daily time scale can differ significantly from values of α estimated over much longer time scales (months).

[4]Note that in our setting $1000 \cdot \delta t$ equals the average number of video downloads in progress.

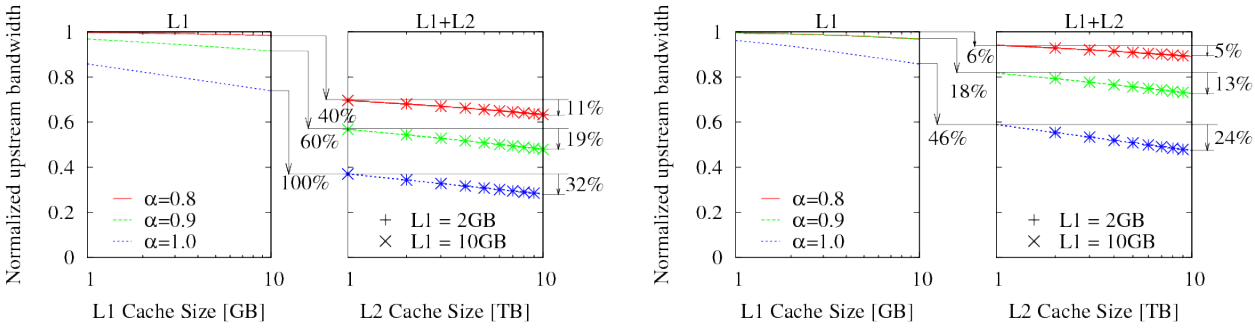

Fig. 3. Comparison between the performance of a standard cache and our two-layer cache: Normalized bandwidth in the (a) current and (b) future scenarios.

between analysis and simulation confirms the validity of the system insensitivity property to the chunk inter-arrival time.

B. Bandwidth performance

We contrast our proposed two-layer cache architecture against a conventional single-layer cache. As main performance index, we evaluate the bandwidth externally requested by the router as computed in (7). We set $p_{m,c}$ and $q_{m,c}$ by experimental characterization of YouTube users reported in [18]. In particular $p_{m,c}$ decreases linearly from 1 to about 0.2, and we furthermore stress the system by selecting $q_{m,c}$ uniformly with $\sum_c q_{m,c} = 2$. This corresponds to assume that users pause, rewind or fast-forward any video two times on average (conservatively overestimating the occurrence of VCR-events with respect to [18]).

We limitedly consider the most challenging $M = 5 \cdot 10^8$ catalog size to gather very conservative performance. We report performance of our system in current and future streaming rate scenarios, in Fig. 3-(a) and 3-(b), respectively. Left subplots report the normalized (to the total demand) bandwidth externally requested by a single-layer ICN router equipped with a DRAM of size in the 1GB to 10GB range. Right subplots report the normalized bandwidth of a two-layer ICN cache. In the hierarchical system, we let L2 size vary from 1 TB to 10 TB, and as L1 memory is more costly, we consider two different configurations with either 2GB or 10GB DRAM equally partitioned between SoV and Swap (cost reduction mainly comes from the shrinking SRAM index). We recall a substantial difference between configurations employing a L1 Swap size of 2 GB vs 10 GB: the former, indeed, can sustain at most 4-4.5 Gbps of aggregate video (and may be suitable for the network periphery), while the latter can scale up to 20-24 Gbps (higher level).

As expected, our two-layer architecture significantly outperforms the monolithic cache architecture, leading to significant bandwidth savings. More in details, bandwidth savings with respect to a single-layer 10GB cache in the current scenario depicted in Fig. 3-(a) range from 40% ($\alpha = 0.8$) up to 100% ($\alpha = 1.0$) even when only 1TB is devoted to L2 storage. Increasing the layer L2 cache to 10 TB yields an extra gain from about 11% ($\alpha = 0.8$) to 32% ($\alpha = 1$). Additionally, while in the future scenario the benefits of a single-layer cache almost completely vanish for $\alpha < 1$, our architecture still leads to sizeable reduction of the upstream bandwidth (about 10% for the very unfavorable $\alpha = 0.8$ scenario and 30% for $\alpha = 0.9$). Under realistic $\alpha = 1$ popularity settings [29], we expect the hierarchical cache system to reduce current (future) ISP bandwidth by 75% (50%) compared to a cacheless scenario.

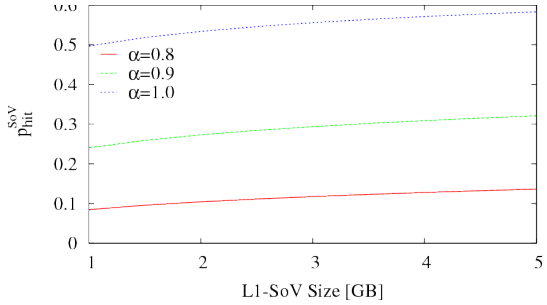

Fig. 4. Cache hit probability at L1-SoV.

Finally, provided that the Swap size satisfies minimal requirements, we observe that L1 dimension is not critical for what concerns the upstream bandwidth: in the hierarchical L1+L2 system, performance corresponding to different L1 configurations are hardly distinguishable. L1 dimensioning has, instead, an impact on the latency incurred by first chunk requests, as latency experienced by the first interest directly relates to the hit probability experienced by SoV, that we report in Fig. 4. Indeed, consider that interest packets served by L1-SoV cache are served with negligible latency (the time to access a DRAM is about 50ns), with respect to chunks that are to be found in the ICN network (the time to retrieve a chunk is in this case dominated by network propagation delay).

C. Trace-driven experiment

To gather an idea of realistic performance such a system could achieve in an operational deployment, we resort to trace-driven simulation. We collect a trace of YouTube video requests at a PoP of a large Italian ISP, offering Internet access to residential customers through ADSL and FTTH technologies. The trace has been extracted by means of Tstat [31], a flow level logger capable of DPI classification and advanced monitoring functionalities. During a period of 35 days, from March 20th to April 25th 2012, we recorded 3.8M requests for 1.76M videos coming from 31K distinct IP addresses.

In our trace-driven simulations, we first fill caches during a warmup phase by cyclically repeating requests from the trace.

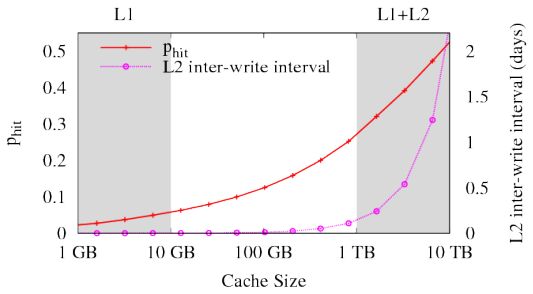

Fig. 5. Hit probability and L2 inter-write interval for the trace-driven experiment.

Then we replay the trace once more to get hit statistics.

Fig. 5 reports, on the left y axes, the average hit probability of the system. The plot highlights two regions: the left region corresponds to cache sizes typical of a single-layer system, whereas the right one corresponds to the L2 cache size (which essentially drives the overall router performance) of a two-layer system. Trace driven experiments confirm that, in the range of values of L2 cache size which are made feasible by our architecture (i.e., 1-10 TB), we achieve significant hit probabilities (i.e., 27%-57%).

Fig. 5 also reports, on the right y axes, the average interval ΔT_{write} between two consecutive write operations over the same physical block in the L2 cache, computed as:

$$\Delta T_{write} \approx \frac{C_B}{\ln(C_B)\Lambda_B(1-p_{\text{hit}}^{\text{L2}})}$$

where C_B is the cache size expressed in number of blocks, whereas Λ_B is the aggregate rate at which batches are requested. The logarithmic factor in the above formula accounts for the fact that blocks are written in random SSD positions, deviating thus from an equal usage of each block. We conclude that, in an edge scenario like the one considered in our experiment, the frequency of write operations on the SSD disk is not really an issue (being two consecutive writes in the 1-10TB range spaced by days, it would take over 30 years for 100,000 write cycle to worn out a 1TB SSD cache).

VI. Conclusions

We design, model and analyze a two-layer caching system that exploits memory technology diversity combining a multi-TB L2 cache to a multi-Gbps L1 cache. Keys to our remarkable performance are (i) exploiting ICN chunks by triggering prefetching, (ii) moving SSD bottleneck from access time to external data rate by prefetching batches of chunks.

As brilliantly pointed out in [3] "changing the overall network architecture in order to tame the exponentially growing world of content with the logarithmic sword of caching seems a classical example of taking a knife to a gunfight: it may make for a great story, but it won't end well." Our work precisely goes in this direction and, by climbing several orders of magnitude of the cache size, it reequilibrate odds in the above gunfight: otherwise stated, if guns are unloaded and the sword is long enough, it makes sense to carry it around.

Advertisement

The research leading to these results has received funding from the European Union under the KIC EIT ICT Labs Project Smart Ubiquitous Contents (SmartUC).

References

[1] I. Psaras, R. G. Clegg, R. Landa, W. K. Chai, and G. Pavlou, "Modelling and Evaluation of CCN-Caching Trees," *IFIP Networking*, 2011.
[2] G. Carofiglio, M. Gallo, L. Muscariello, and D. Perino, "Modeling Data Transfer in Content-Centric Networking," in *ITC*, 2011.
[3] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan, and J. Wilcox, "Information-centric networking: seeing the forest for the trees," in *ACM HotNets-X*, 2011.
[4] S. Arianfar and P. Nikander, "Packet-level Caching for Information-centric Networking," in *ACM SIGCOMM, ReArch Workshop*, 2010.
[5] D. Perino and M. Varvello, "A reality check for content centric networking," in *ACM SIGCOMM, ICN Workshop*, 2011.
[6] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *Communications Magazine, IEEE*, vol. 50, pp. 26 –36, july 2012.
[7] V. Jacobson, D. K. Smetters, N. H. Briggs, J. D. Thornton, M. F. Plass, and R. L. Braynard, "Networking Named Content," in *ACM CoNEXT*, 2009.
[8] H. Yuan, T. Song, and P. Crowley, "Scalable ndn forwarding: Concepts, issues and principles," in *ICCCN*, 2012.
[9] W. You, B. Mathieu, P. Truong, J.-F. Peltier, and G. Simon, "Realistic storage of pending requests in content-centric network routers," ICC, 2012.
[10] W. You, B. Mathieu, P. Truong, J.-F. Peltier, and G. Simon, "Dipit: A distributed bloom-filter based pit table for ccn nodes," in *ICCCN*, 2012.
[11] M. Varvello, D. Perino, and L. Linguaglossa, "On the Design and Implementation of a wire-speed Pending Interest Table," in *NOMEN*, 2013.
[12] H. Yuan, B. Wun, and P. Crowley, "Software-based implementations of updateable data structures for high-speed url matching," in *ANCS*, 2010.
[13] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan, "Fawn: A fast array of wimpy nodes," in *ACM SIGOPS'09*.
[14] A. Badam, K. Park, V. S. Pai, and L. L. Peterson, "Hashcache: Cache storage for the next billion.," in *USENIX NSDI*, 2009.
[15] H. Che, Y. Tung, and Z. Wang, "Hierarchical web caching systems: Modeling, design and experimental results," *IEEE Journal on Selected Areas in Communication (JSAC)*, vol. 20, no. 7, pp. 1305–1314, 2002.
[16] N. Laoutaris, S. Syntila, and I. Stavrakakis, "Meta Algorithms for Hierarchical Web Caches," in *IEEE ICPCC*, 2004.
[17] A. Aggarwal, B. Alpern, A. Chandra, and M. Snir, "A model for hierarchical memory," in *ACM Annual Symp. on Theory of Computing*, 1987.
[18] A. Finamore, M. Mellia, M. Munafo, R. Torres, and S. Rao, "Youtube everywhere: Impact of device and infrastructure synergies on user experience," in *ACM IMC*, 2011.
[19] https://support.netflix.com/en/node/306.
[20] http://imaginlab.fr/blog-en/?p=480.
[21] http://www.nngroup.com/articles/computer-screens-getting-bigger/.
[22] http://www.seagate.com/internal-hard-drives/enterprise-hard-drives/hdd/enterprise-capacity-3-5-hdd/.
[23] http://www.seagate.com/internal-hard-drives/enterprise-hard-drives/ssd/x8-accelerator/.
[24] http://ramsan.com/products/pcie-storage/ramsan-70.
[25] http://www.named-data.net/.
[26] M. Cornwell, "Anatomy of a solid-state drive.," *Commun. ACM*, vol. 55, no. 12, pp. 59–63, 2012.
[27] C. Fricker, P. Robert, and J. Roberts, "A versatile and accurate approximation for lru cache performance," ITC'12, 2012.
[28] M. Cha, H. Kwak, P. Rodriguez, Y. Ahn, and S. Moon, "I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system," in *ACM IMC*, 2007.
[29] S. K. Fayazbakhsh, Y. Lin, A. Tootoonchian, A. Ghodsi, T. Koponen, B. M. Maggs, K. C. Ng, V. Sekar, and S. Shenker, "Less pain, most of the gain: Incrementally deployable icn," in *ACM SIGCOMM*, 2013.
[30] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "Youtube traffic characterization: a view from the edge," in *ACM IMC*, pp. 15–28, 2007.
[31] A. Finamore, M. Mellia, M. Meo, M. Munafo, and D. Rossi, "Experiences of internet traffic monitoring with tstat," *IEEE Network*, 2011.