Probabilistic In-Network Caching for Information-Centric Networks *

Ioannis Psaras, Wei Koong Chai, George Pavlou
Dept. of Electrical & Electronic Engineering, University College London
WC1E 7JE, Torrington Place, London, UK
{i.psaras, w.chai, g.pavlou}@ucl.ac.uk

ABSTRACT

In-network caching necessitates the transformation of centralised operations of traditional, overlay caching techniques to a decentralised and uncoordinated environment. Given that caching capacity in routers is relatively small in comparison to the amount of forwarded content, a key aspect is balanced distribution of content among the available caches. In this paper, we are concerned with decentralised, real-time distribution of content in router caches. Our goal is to *reduce caching redundancy* and in turn, make more efficient utilisation of available cache resources along a delivery path.

Our in-network caching scheme, called *ProbCache*, approximates the *caching capability* of a path and caches contents probabilistically in order to: *i)* leave caching space for other flows sharing (part of) the same path, and *ii)* fairly multiplex contents of different flows among caches of a shared path.

We compare our algorithm against universal caching and against schemes proposed in the past for Web-Caching architectures, such as Leave Copy Down (LCD). Our results show reduction of up to 20% in server hits, and up to 10% in the number of hops required to hit cached contents, but, most importantly, reduction of cache-evictions by an order of magnitude in comparison to universal caching.

Categories and Subject Descriptors

C2.1 [**Computer-Communication Networks**]: Network Architecture and Design - Distributed Networks

Keywords

In-Network Caching, Information-Centric Networks

*The research leading to these results was supported by the EU FP7 COMET project, under Grant Agreement 248784.

ICN'12, August 17, 2012, Helsinki, Finland.

1. INTRODUCTION

Naming content objects directly, instead of their respective end-hosts, gives the opportunity to identify content objects as they travel from source to destination [16, 13, 12]. In turn, given that the network transfers *named* objects (instead of unidentifiable *data containers, i.e.*, IP packets), these objects can be cached *in the network* and be forwarded to subsequent users interested in the same content [1, 2].

In-network caching has therefore emerged as a distinct research field in the context of *Information-Centric Networks* (ICN). In-network caching exhibits fundamental differences from overlay web-caching [5], or hierarchical and co-operative caching approaches [8, 11] and poses new challenges [20, 19]. For instance, past research considered mainly caching of whole files (with a few exceptions [23]) as well as administration of their placement [14] and location [17] by centralised entities, *e.g.*, DNS and HTTP redirection. Centralised administration of content placement gives the opportunity to control and manage network resources better at the cost of: *i)* increased communication overhead to update the content location database, and *ii)* reduced flexibility in terms of available cache locations.

In contrast, Information-Centric Networking enables caching of addressable content *chunks* [13, 23] in *every* cache-equipped network device [21] and replacement of cached chunks at line-speed [3]. Although, this operation increases the availability of cache locations [22] it renders prohibitive the process of updating logically-centralised content location databases with the exact location of cached contents. This decentralised, location-independent operation alters many of the basic features of past overlay caching techniques, *e.g.*, content-to-cache allocation [14], while it invalidates the applicability of some others, *e.g.*, content placement based on fixed overlay topologies of caches and servers [17].

In this paper, we are concerned with cache management operations that have to be adjusted to fit in a completely decentralised and unco-ordinated environment. We focus on the fair sharing of the available cache capacity of *a path* among the content flows that use part of this path per unit time. In other words, we focus on the allocation of the available cache capacity along a *path* of caching entities among different content flows. As a starting point we intuitively observe that caching *every* content in *every* cache-enabled device along the delivery route (an operation implicitly supported in [13]), inherently causes huge *caching redundancy*. Subsequently, our goal is to reduce caching redundancy and make more efficient use of available cache resources, in order to reduce overall network utilisation and potentially increase

user-perceived quality. Indeed, our initial investigation of selective caching policies based on node centrality metrics, [7], shows very promising results on this direction.

To achieve our goal, we approximate the *caching capability* of a given path *per unit time* (Section 2.2) and we design *ProbCache*, a *probabilistic algorithm* for distributed content caching along a path of caches (Section 3). Our results suggest that there is a lot of space for resource management optimisation of in-network caching policies, given that appropriate content multiplexing rules are in place (Section 4).

We use the terms "router" and "cache" interchangeably to refer to cache-enabled network devices [3]; it should be noted that our approach does not require every router to be cache-enabled, but it will work in hybrid architectures as well. Furthermore, we refer to content "packets", "messages" and "chunks" interchangeably to refer to the *cacheable unit*, which is not necessarily of similar size to an IP packet. In fact, we leave open the actual size of the cacheable unit which is yet to be defined by the ICN research community. We highlight that the concepts and algorithms proposed here are *cache unit-* as well as *architecture-agnostic* and would apply to almost any ICN environment [6, 13, 9, 4].

We differentiate between two types of *redundancy*, namely, *network traffic redundancy* [2] and *caching redundancy* [14, 11]. Caching has been traditionally used to reduce traffic redundancy [1, 5]. Assuming a set of available caches, as opposed to a single proxy cache, redundancy can still exist between cached contents in different locations, something that has also been investigated in the past for overlay caching schemes [14, 11]. However, and as mentioned earlier, overlay schemes require some form of co-operation between the caches themselves and/or between the caches and a central management entity. In contrast, in the case of in-network caching, management has to happen in an unco-ordinated, unco-operative fashion. In this paper, we consider each path of caching entities as a pool of caching resources; we try to find optimal ways of distributing content in these caches in order to eliminate *caching redundancy* and in turn, reduce *traffic redundancy*. Our results show that careful content flow multiplexing in caches can achieve up to 20% more cache hits. Surprisingly, this translates to one order of magnitude reduction in terms of traffic redundancy.

2. SYSTEM MODEL AND ASSUMPTIONS

We argue that *in-network cache management has to take into account the approximate cache capacity of the path of caches and the estimated amount of traffic that these caches serve per unit time, in order to make decisions on whether to cache incoming contents or not.* In Section 2.1, we make assumptions that help us approximate the cache capacity of a given path and in Section 2.2, we present our design principles.

2.1 Assumptions on Caching Technologies

By definition, *caching* is different to *storage*, both in networks and in computer systems, in that caching keeps contents stored *for a specific amount of time and not indefinitely*, as in storage. Therefore, the size of a cache is a relative factor, which cannot stand on its own, but instead has to be linked to the amount of time that a given content is cached for. We, therefore, associate the cache size with the traffic that the corresponding router serves per second. Our cache size unit is the *number of seconds worth of traffic cached* in a given router and depends on the speed of the outgoing links of the router in question.

Link Name	Link Speed	1-sec traffic	traffic in 10GBs
OC-24	1,2 Gbps	$\sim$ 0.15 GBs	$\sim$ 64 secs
OC-48	2,4 Gbps	$\sim$ 0.31 GBs	$\sim$ 32 secs
OC-192	9,9 Gbps	$\sim$ 1.25 GBs	$\sim$ 4 secs
OC-768	39,8 Gbps	$\sim$ 5 GBs	$\sim$ 2 secs
OC-1536	79,6 Gbps	$\sim$ 10 GBs	$\sim$ 1 sec
OC-3072	159,2 Gbps	$\sim$ 20 GBs	$\sim$ 0.5 secs

Table 1: Link Speeds and Caching Properties

One important question then is: *"For how long can we afford to cache contents in a given router?"*. Furthermore, given that in this study we are concerned with *paths* of caches and not with single-caches only, another important question is: *"For how long do we need to cache contents in a given path in order to minimise redundant traffic and maximise gain?"*. Our reasoning for answering these questions is as follows:

- Today's memory access technologies guarantee *line-speed* access to DRAM chips of up to 10GBytes at a reasonable price [3]. This means that a 5GByte-long cache behind a 40Gbps link[1] can safely be assumed to hold contents for one second (see also Table 1). *Without loss of generality, we assume that each cache along a path has sufficient memory to cache contents in a DRAM chip for at least one second* (see third column in Table 1).

- Authors in [2] show up to 60% bandwidth savings by redundant traffic elimination *within the first 10 seconds after the original transmission*, in some enterprise networks. We associate redundant traffic, *i.e.*, subsequent requests for the same content, with the aforementioned result. That is, we consider, without loss of generality, that any content should be kept in any one of the path's caches for a *target time window*, T_{tw}, of 10 seconds.

Both the above settings are relatively arbitrary and can change in the future, but these values are a good starting point based on today's technology. In addition, our concepts and algorithms presented next are still applicable should these values change.

2.2 System Model

The path from source to destination in our topology (Fig. 1) comprises n routers, where router r_i has N_i cache slots, each able to hold one addressable content chunk; based on the discussion above, we assume that N_i slots can hold one second worth of traffic. Our notation is given in Table 2.

Path Cache Capacity. The *caching capacity* of our path of caches is $\sum_{i=1}^{n} N_i$, in terms of memory, which amounts to n seconds worth of traffic cached along the path.

[1]To the best of our knowledge, operators to date use links of up to 40 Gbps, while some operators plan to update a limited number of their links to 100 Gbps: http://www.prnewswire.com/news-releases/verizon-first-service-provider-to-announce-100g-deployment-on-us-network-118891754.html

SYMBOL	MEANING
n	Number of caches on the path
N_i	Cache memory in r_i - holds 1-sec worth of traffic
T_{tw}	Target Time Window (set to 10 secs here)
TSI, c	Time Since Inception (*Request* Message Header): Hop-Distance from Client, Value range: 1 *to* n
TSB, x	Time Since Birth (Content Message Header): Hop-Distance from Server, Value range: 1 *to* n

Table 2: Model Notation

Path Cache Capability. Given that our target time window is T_{tw} seconds worth of traffic cached along a given path, the *caching capability* of an n-long path, as a fraction of the required capacity for T_{tw} seconds, is $\frac{\sum_{i=1}^{n} N_i}{T_{tw} N}$, where N is the average cache size along the path. We revisit the issue of average cache size in the next section.

Symmetric Paths. Request and Content messages follow the same route, similarly to [16], [6], [13], [15].

Path Length Monitoring. Similar to the TTL field included in IP packets, our design requires that ICN *request message* headers include the *Time Since Inception* (TSI) field and *content message* headers include both the TSI and the *Time Since Birth* (TSB) fields. Every router increases the TSI value of request packets by one. The content source attaches the TSI value that it sees on the request message to the content message. Every router increases the TSB value of the content message by one. Hence, during the content message's journey to the client, the TSI value is a fixed value and denotes the path-length of this specific content flow, while the TSB value denotes the number of hops that the content message has travelled so far[2].

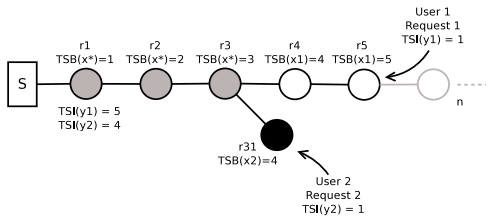


Figure 1: Design Topology

3. PROBABILISTIC IN-NETWORK CACHING

We approach the problem of content placement within a system of caches from the *path caching capability* point of view. In particular, *each router, based on the amount of traffic that it has to serve per unit time, indirectly approximates the number of copies of incoming contents that the (rest of the) path can accommodate.* This value is the $TimesIn$ factor (Section 3.1). Based on this indication and on *the router's distance from the user*, which we call $CacheWeight$ (Section 3.2), each router probabilistically caches contents as they travel along the path (Section 3.3). We consider the example topology of Fig. 1, where two users are connected five and four hops away from the server, respectively.

[2]In case of a cache hit, the TSI and TSB values are treated as if the cache is the actual source, that is, the TSI value of the content packet is replaced by that of the *Request* packet, while the TSB is set to 1.

3.1 Estimating Caching Capability of Paths

The total cache capacity of the path in Fig. 1 is $\sum_{i=1}^{n} N_i$, where $n_1 = 5$ for $Request_1$ and $n_2 = 4$ for $Request_2$. Grey circles denote the caches that have to be shared between the two users, while white and black circles denote caches used exclusively by Users 1 and 2, respectively.

The number of times that the path can afford to cache this packet is reflected in the $TimesIn$ factor, whose calculation takes place as follows:

$$TimesIn(x) = \frac{\sum_{i=1}^{c-(x-1)} N_i}{T_{tw} N_x} \quad (1)$$

where c is the *Time Since Inception* (TSI) value and x is the *Time Since Birth* (TSB) value that the router sees in the header of the content message (Table 2). For example, content messages traveling through router r_2 to fulfil $Request_1$, in Fig. 1, will have $TSI = 5$ and $TSB = 2$, while contents for $Request_2$ will have $TSI = 4$ and $TSB = 2$. The sum in Eq. 1 incorporates the result of the subtraction of TSI minus TSB (or $c - (x - 1)$ in Eq. 1) to account for the *remaining* caches only, instead of the total number of caches from the content source to the client.

3.2 Weight-based Caching

We argue that in order to achieve fair resource (in our case cache) allocation in a distributed environment, each content flow has to take into account other content flows sharing the same path (grey circles in Fig. 1). Hence, to decide *where* to cache the number of copies that $TimesIn$ indicated, we use the *Cache Weight* factor:

$$Cache\ Weight(x) = \frac{x}{c} \quad (2)$$

where x is the TSB value of the packet header and c is the TSI value. We note that the TSI value is fixed during the content packet's journey from the source to the client, while the TSB value is increasing for each router the packet traverses; hence, $CacheWeight \rightarrow 1$ as the packet is getting closer to its destination. This is a desirable system property considering path-diversity, in terms of number of hops, between clients and sources of different content flows.

3.3 Building ProbCache

ProbCache is the product of $TimesIn$ and $CacheWeight$. Each router caches incoming chunks with probability $ProbCache$, depending on their TSI and TSB values.

$$ProbCache(x) = \underbrace{\frac{\sum_{i=1}^{c-(x-1)} N_i}{T_{tw} N_x}}_{TimesIn} \times \underbrace{\frac{x}{c}}_{CacheWeight} \quad (3)$$

The *Cache Weight* factor increases the probability of a content being cached closer to its destination. This way, we achieve *fair content flow multiplexing* between contents that travel to different destinations in terms of path length. For example, contents for User 1 in Fig. 1 should be cached inversely proportionally to User 1's distance from the server, *i.e.*, in (white) routers r_4, or r_5, in order to leave (grey) routers $r_1 - r_3$ for clients travelling shorter paths to cache their contents. This is in accordance with our previous findings in [22] that contents tend to be cached for longer towards the edge of the network.

NOTE: To calculate the $TimesIn$ factor each router has to conjecture on the size of the rest of the routers on the path. However, given that we do not know what amount of memory each router will have, or if backbone routers, for instance, will have bigger caches than edge-network routers, we make the following simplifying assumption. *Each router assumes that all other routers on the path have the same amount of cache as it has got.* Even in a random-size cache deployment scenario, this assumption serves our purposes well. That is, a router with a big cache, compared to the caches along the path, will be caching contents with higher probability, while a router with a small cache will experience the opposite effect (Eq. 1). This is a desirable system property which alleviates the effect of unknown cache sizes, but at the same time guarantees fair load distribution among nodes with diverse caches. We relax the assumption of homogeneous cache sizes later on and show that although our simplifying assumption does not harm the performance of *ProbCache* in heterogeneous cache size environments, it fails to exploit extra caching resources (Section 4.2).

4. PERFORMANCE EVALUATION

We test our algorithm in a custom-built simulator, where we use *Least Recently Used* (LRU) caches. Given that the ultimate goal of *ProbCache* is to manage caching resources more efficiently, by reducing cache redundancy, the straightforward metric of interest is the *reduction of Server Hits*. Furthermore, the gain from serving user requests from intermediate caches, instead of travelling to the origin server, depends on the number of hops that the request travels before it eventually hits cached contents. Clearly, as the number of hops increases, the overall gain decreases. To measure this gain, we also monitor the *Hop Reduction Ratio*.

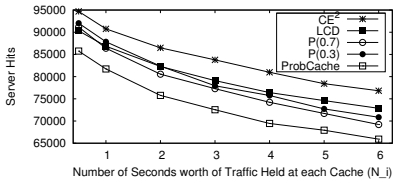
We use binary tree topologies; set the exponent of the Zipf distribution of content popularity to 0.8 to capture the case of fairly unpopular content [25]; and compare the performance of *ProbCache* against: *i)* the universal caching approach proposed in [13], a scheme that we call *Cache Everything Everywhere* (CE^2), *ii)* the Leave Copy Down (LCD) [18] algorithm proposed in the past for overlay caching topologies[3], *iii)* a probabilistic algorithm that caches with probability $p = 0.7$ at every cache and *iv)* a probabilistic algorithm that caches with probability $p = 0.3$.

4.1 Scenario 1: Path Resource Management

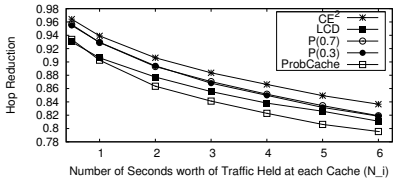
We use a 6-level binary tree topology (127 nodes in total). The root node represents the server node and we configure requests to come from the last two levels of the tree, assuming that individual users are not connected to backbone network routers; we note however that results are similar in case of users attached to all routers. We generate a total of 100,000 requests, to allow enough time for the system to reach a steady-state. As noted earlier, our cache-size unit is the *number of seconds worth of traffic cached in a given router*. In our first experiment, we assume homogeneous caches and we evaluate the algorithms as we increase each router's cache capacity from one to six seconds (Fig. 2).

The Server Hits performance difference balances around a reduction of approximately 12-15% (12,000-15,000 Server Hits less) for *ProbCache* against CE^2 and around 7-10% compared to the rest of the algorithms evaluated here (Fig. 2(a)).

The *Hop Reduction* difference is roughly 8-10% against the CE^2 scheme, while it is smaller compared to the rest of the algorithms and especially compared to LCD (up to 3%). However, considering short path lengths (up to 6 hops) and the operational properties of both CE^2 and LCD, whose design targets bringing content as close to the end-user as possible, we argue that even this small performance difference unveils better exploitation of storage resources in merit of *ProbCache*. To validate this claim further, we trace the average number of: *i)* Cache Hits, *ii)* Received Requests, and *iii)* Cache Evictions per tree level. The size of the cache is set to three seconds worth of traffic at each cache.



(a) Server Hit Saving



(b) Hop Reduction Ratio

Figure 2: Scenario 1: Homogeneous Caches

We observe, although do not present the related results here due to space limitations, disproportionate differences between the number of Cache Hits and the corresponding number of Received Requests in case of *ProbCache*. That is, *ProbCache* has more Cache Hits and less Received Requests than the rest of the algorithms. This difference in Cache Hits and Received Requests owes to the efficient resource management of *ProbCache*, which results in contents staying in caches for longer. To substantiate our claims, we plot in Fig. 3 the average number of evictions per tree-level, where the smaller the tree level, the closer to the server this level is. The very big difference in terms of cache evictions validates our claim that per-path, distributed resource management results in more contents staying in the caches for longer and therefore, getting more cache hits. This huge reduction in cache evictions in Fig. 3 reflects the reduction in terms of traffic that flows through the network and subsequently, redundant traffic elimination.

4.2 Scenario 2: Heterogeneous Cache Sizes

We proceed to relax the assumption of homogeneous caches. In particular, we expect that as research in the area matures, caches will be sized according to a specific norm or a set of guidelines. We conjecture three potential norms: *i)*

[3]According to LCD [18], every request for a specific content causes the content to be copied one hop closer to the user, or one level down in the cache hierarchy. If a request for a content is received after the content has been evicted from the cache, according to the LRU policy, the content has to be retrieved from another cache up-the-hierarchy, or from the origin server.

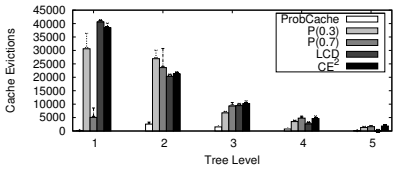


Figure 3: Average Cache Evictions per Tree Level

larger caches are deployed towards the core of the network, where servers reside, *ii)* larger caches are deployed towards the edges of the network, where users are connected, *iii)* all caches have roughly similar sizes (homogeneous cache scenario presented in the previous section).

We use the same topology as before, set cache sizes according to the formula: $N_i \times TSB$ (see Table 2) and observe the performance difference in case of heterogeneous caches. That is, if a core router caches one second worth of traffic and we assume larger caches towards the edges of the network (which we denote as $c(ore) \rightarrow E(dge)$), then the edge router of a six-hop path will cache six seconds worth of traffic. The capacity of the path in that case is $c \rightarrow E : \sum_{i=1}^{c-(x-1)}(c-i)N_i$. The opposite applies for deployments where larger caches are deployed in the core of the network (which we denote as $C(ore) \rightarrow e(dge)$); the capacity of the path in this case is $C \rightarrow e : \sum_{i=1}^{c-(x-1)} iN_i$.

We modify the *TimesIn* factor based on the above path capacities for each case in Eqs. 4 and 5, respectively and call the new algorithm *ProbCache+*. The corresponding cache capacity calculations along a six-hop path for Eqs. 1, 4 and 5 are shown in Fig. 4. We use the default algorithm (*i.e.*, Eqs. 1 and 3) as our benchmark for comparisons, and for clarity, we compare the performance of *ProbCache+* against CE^2 [13] and LCD [18] only.

$$TimesIn_{(C\rightarrow e)}(x) = \frac{\sum_{i=1}^{c-(x-1)} iN_i}{T_{tw}N_x} \quad (4)$$

$$TimesIn_{(c\rightarrow E)}(x) = \frac{\sum_{i=1}^{c-(x-1)} (c-i)N_i}{T_{tw}N_x} \quad (5)$$

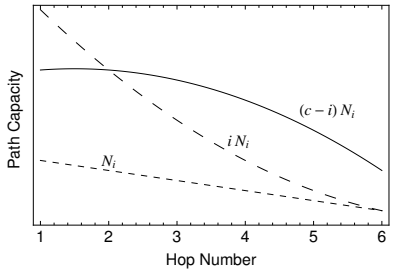
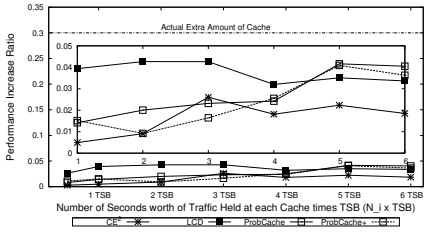


Figure 4: Path Capacity calculation that takes place at each router along a 6-hop Path: the N_i curve is for homogeneous caches, hence the capacity is given by $c \rightarrow e : \sum_{i=1}^{c-(x-1)} N_i$; the iN_i curve is for larger core caches, hence $C \rightarrow e : \sum_{i=1}^{c-(x-1)} iN_i$; the $(c-i)N_i$ curve is for larger caches towards the edge, hence $c \rightarrow E : \sum_{i=1}^{c-(x-1)}(c-i)N_i$.

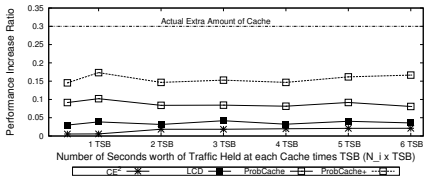
In Fig. 5 we present the ratio of each protocol's cache hits performance in case of extra cache added in the core (Scenario 2.1) or the edge (Scenario 2.2) over the performance in case of homogeneous caches (Scenario 1).

As noted, *ProbCache* is the original algorithm in Eq. 3, hence this algorithm does not take into account the extra caching capacity; in contrast, *ProbCache+* incorporates Eq. 4 in Scenario 2.1, Fig. 5(a), and Eq. 5 in Scenario 2.2, Fig. 5(b). In both cases, we observe that both CE^2 and LCD fail to utilise the extra available cache capacity; that is, their performance increases by less than 5%, although the extra cache capacity we added is 30% compared to Scenario 1 (Section 4.1). We see that this is the case too with *ProbCache* and *ProbCache+* in case of extra cache capacity at the core of the network (*i.e.*, Fig. 5(a))[4].

Interestingly, and in contrast to recent results reported in [26], where the authors show that cache size heterogeneity does not improve the caching performance in ICNs, we find that both *ProbCache* and especially *ProbCache+* utilise a good amount (up to 18%) of the extra caching capacity added towards the edge of the network in Scenario 2.2, Fig. 5(b). The overall performance difference in that case is between 20-25% compared to CE^2 and LCD, if we take into account the absolute results reported in Section 4.1, Fig. 2(a). We argue that this is due to more efficient resource utilisation of the proposed algorithm and better content multiplexing in caches, something that does not constitute a design goal for either CE^2 or LCD.



(a) Sc. 2.1: Larger Cache at Core ($C \rightarrow e$)



(b) Sc. 2.2: Larger Cache at Edge ($c \rightarrow E$)

Figure 5: Scenario 2: Heterogeneous Caches – y-axis measures the performance difference ratio compared to the homogeneous scenario presented in Fig. 2 – the inner part of Fig. 5(a) is a zoom-in to the lineplots of the same figure.

Summarising, we argue that these are the properties of *in-network caching* that the research community needs to investigate further, in order to exploit this inherent capability of ICNs for in-network caching. We have tested our algorithm in scale-free topologies with multiple servers and multiple replicas of contents. Our results show even greater

[4]We note that in Fig. 5 we present the performance increase of the protocols compared to the previous scenario. This means that the performance of the algorithms in absolute terms is similar to the one presented in Fig. 2.

differences in terms of redundant traffic elimination in favour of *ProbCache*. In case of CE^2 this owes to inefficient resource utilisation, while in case of LCD it owes to its inherent design to fit to directed graphs only, where the direction from servers to clients is a fixed hierarchy. In contrast, *ProbCache* exploits the knowledge provided by TSI and TSB and multiplexes content flows according to their respective path lengths.

5. RELATED WORK

Several recent studies have focused specifically on the properties of a network with in-network caches (*e.g.*, [20], [25], [24]). In [25], the authors provide a comprehensive performance evaluation of in-network caching taking into account several parameters, such as request distributions, the catalog size, and cache replacement policies. They conclude that content popularity is (by far) the most important parameter of all. Along the same lines, authors in [10] investigate the impact of traffic mix on the caching performance of a two-level cache hierarchy. They conclude that VoD content should be cached towards the edge of the network, while other types of content should be stored in large discs towards the core.

In a recent study [26], the authors show that having heterogeneously sized caches does not improve overall performance. We argue that this owes to the assumption of ubiquitous caching, that is, caching chunks in all routers along the content delivery path. As we have shown both in this paper (see Section 4.2, Fig. 5) and in our recent study [7], by caching a limited number of copies of a chunk in selected caches in the network, we achieve significantly higher gains. In particular, in [7], we show that the centrality of nodes in a given network topology gives valid evidence of which nodes are within the most number of paths. In turn, caching in those nodes increases the overall performance [7].

6. CONCLUSIONS

We have argued that caching named chunks in network routers' DRAM memory, as opposed to caching large objects or files in proxy disks, calls for reconsideration of past approaches to caching. That is, in-network caching in ICNs has to happen in an uncoordinated and distributed fashion. We have proposed *ProbCache*, an algorithm that approximates the capability of paths to cache contents, based on path lengths, and multiplexes content flows accordingly. The ultimate goal of *ProbCache* is to utilise resources efficiently, reduce *caching redundancy* and in turn, *network traffic redundancy*. We have considered both homogeneous and heterogeneous cache sizes and have adjusted *ProbCache* to fit in both environments.

We report savings of up to 20% in server hits; 7-8% in the number of hops to hit cached contents; and reduction by an order of magnitude in cache evictions, which directly translates to network traffic redundancy elimination by the same proportion.

7. REFERENCES

[1] A. Anand and et al. Packet caches on routers: the implications of universal redundant traffic elimination. In *ACM SIGCOMM*, pages 219–230, 2008.

[2] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramjee. Redundancy in network traffic: findings and implications. In *SIGMETRICS '09*.

[3] S. Arianfar, P. Nikander, and J. Ott. On content-centric router design and implications. In *ReArch Workshop*, volume 9, page 5. ACM, 2010.

[4] J. M. Batalla, A. Beben, and Y. Chen. Optimization of the decision process in NEtwork and Server-aware algorithms. In *Proc. of IEEE Networks 2012*, 2012.

[5] L. Breslau and et al. Web caching and zipf-like distributions: Evidence and implications. In *In INFOCOM*, pages 126–134, 1999.

[6] W. K. Chai and et al. Curling: Content-ubiquitous resolution and delivery infrastructure for next-generation services. *IEEE Communications Magazine*, 49(3):112–120, 2011.

[7] W. K. Chai, D. He, I. Psaras, and G. Pavlou. Cache 'Less for More' in Information-centric Networks. In *Proc. of IFIP Networking*, 2012.

[8] H. Che, Z. Wang, and Y. Tung. Analysis and Design of Hierarchical Web Caching Systems. In *INFOCOM*, pages 1416–1424. IEEE, 2001.

[9] N. Fotiou, G. C. Polyzos, and D. Trossen. Illustrating a publish-subscribe internet architecture. *Journal on Telecommunication Systems, Springer, March*, 2011.

[10] C. Fricker, P. Robert, J. Roberts, and N. Sbihi. Impact of traffic mix on caching performance in a content-centric network. In *IEEE NOMEN*, 2012.

[11] N. Fujita, Y. Ishikawa, A. Iwata, and R. Izmailov. Coarse-grain replica management strategies for dynamic replication of web contents. *Comput. Netw.*, 45(1), 2004.

[12] A. Ghodsi and et al. Naming in content-oriented architectures. In *ACM SIGCOMM ICN Workshop*, pages 1–6, 2011.

[13] V. Jacobson and et al. Networking Named Content. In *ACM CoNEXT '09*, pages 1–12, 2009.

[14] A. A. Jiang and J. Bruck. Optimal content placement for en-route web caching. In *Proc. of IEEE NCA '03*.

[15] K. Katsaros, G. Xylomenos, and G. C. Polyzos. Multicache: An overlay architecture for information-centric networking. *Comput. Netw.*,, 2011.

[16] T. Koponen and et al. A Data-Oriented (and beyond) Network Architecture. *SIGCOMM*, 37(4):181–192, '07.

[17] P. Krishnan, D. Raz, and Y. Shavitt. The cache location problem. *IEEE/ACM Trans. Netw.*, 8(5):568–582, Oct. 2000.

[18] N. Laoutaris, H. Che, and I. Stavrakakis. The lcd interconnection of lru caches and its analysis. *Perform. Eval.*, 63(7), July 2006.

[19] U. Lee, I. Rimac, and V. Hilt. Greening the internet with content-centric networking. *eEnergy*, 2010.

[20] L. Muscariello, G. Carofiglio, and M. Gallo. Bandwidth and storage sharing performance in information centric networking. In *ACM SIGCOMM ICN Workshop*, pages 26–31, 2011.

[21] D. Perino and M. Varvello. A reality check for content centric networking. In *ACM SIGCOMM ICN Workshop*, pages 44–49, 2011.

[22] I. Psaras, R. G. Clegg, R. Landa, W. K. Chai, and G. Pavlou. Modelling and Evaluation of CCN-Caching Trees. In *Proc. of IFIP NETWORKING*, 2011.

[23] E. J. Rosensweig and J. Kurose. Breadcrumbs: Efficient, Best-Effort Content Location in Cache Networks. In *INFOCOM*, 09.

[24] E. J. Rosensweig, J. Kurose, and D. Towsley. Approximate models for general cache networks. In *IEEE INFOCOM*, 2010.

[25] D. Rossi and G. Rossini. Caching performance of content centric networks under multi-path routing. *Technical Report*, 2011.

[26] D. Rossi and G. Rossini. On sizing CCN content stores by exploiting topological information. In *IEEE NOMEN Workshop*, 2012.