Supplementary File

VIII. EXTENDING ProbCache FOR HETEROGENEOUS CACHE SIZES

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 directions: i) 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 size case).

For ease of illustration, we assume that caches are sized according to the tier they belong to. In real deployments, since a router is connected to multiple servers at different (core or backbone network). Although this might not be realistic as a setting in real deployments, since a router is connected to multiple servers at different distances, we consider that similarly to different-tier domains, router caches can be sized according to the tier they belong to.

We conjecture three potential directions:  

1. Equal-sized caches along the path (c → e):

\[ P_{c→e}(x) = \frac{\sum_{i=1}^{c}(x-1)}{T_{tw}N_{x}} \times \frac{N_{i}(c-x+1)}{c} \]  

(7)

2. Larger caches in the core (C → e):

\[ P_{C→e}(x) = \frac{\sum_{i=1}^{c}(x-1)}{T_{tw}N_{x}} \times \frac{N_{i}(c-2x+2)}{c} \]  

(8)

3. Larger caches towards the edge (c → E):

\[ P_{c→E}(x) = \frac{\sum_{i=1}^{c}(x-1)}{T_{tw}N_{x}} \times \frac{(c-i)N_{i}}{c} \]  

(9)

In Fig. 9, we plot the path cache capacity as it is calculated by ProbCache at each node along a six-hop path according to the TimesIn factor of Eqs. 7, 8 and 9.

In [1], we have evaluated the performance of ProbCache (Eqs. 7, 8, 9) in the above three different settings (i.e., equal caches along the path, larger in the core and larger towards the edge). We have used binary-tree topologies and compared the performance of ProbCache against alternative algorithms and caching approaches proposed in the related literature (e.g., universal caching used in [2] and LCD [3]).

In case of homogeneous caches along the path, our results showed a reduction of up to 20% in server hits, and up to 10% in the number of hops taken to hit cached contents, but, most impressively, reduction of cache evictions by an order of magnitude, compared to universal caching. In case of heterogeneous caches along the delivery path, we found that when we deploy more cache capacity in the core of the network the algorithms do not seem to realise and exploit the extra resources available. In contrast, when larger caches are put towards the edge of the network and closer to the end-users, ProbCache exploits most of the extra available resources along the delivery path and improves significantly the overall network performance as perceived by the end-user. We refer the reader to [1] for a more detailed discussion of the results, which we omit here due to space limitations.

Based on the results reported in [1] and briefly summarised above, for the rest of this paper we elaborate on the performance of ProbCache under two cache-size settings, that is, equal-sized caches along the whole path (Eq. 7) and larger caches towards the edge (Eq. 9).

The source is here largely defined as the area where servers are connected (core or backbone network). Although this might not be realistic as a setting in real deployments, since a router is connected to multiple servers at different distances, we consider that similarly to different-tier domains, router caches can be sized according to the tier they belong to.
IX. BEHAVIOUR OF BASIC FUNCTION

A. Maximum Values of Basic Function

In this Section, our intention is to find the number of hops away from the content source that Probcache gets its maximum value for users connected at different points along the path. To achieve this, we initially calculate the derivatives of Eqs. 7 and 9 and then set the derivatives to zero in order to find the \textit{local maxima} of each value. Our calculations give the following for Eqs. 7 and 9:

\[
\frac{dP_{c\rightarrow e}(x)}{dx} = 0 \Rightarrow x = \frac{1 + c}{2} \quad (10)
\]

We plot the derivative of \(P_{c\rightarrow e}(x)\) and of \(P_{c\rightarrow E}(x)\) (Eq. 10) in Fig. 10 for users connected at several different points along the sample 6-hop delivery path. The \(x\)-axis denotes the distance of the client from the source node (i.e., the TSI value - see Table II), while the \(y\)-axis denotes the hop number (i.e., TSB) for which \(Probcache\) gets its highest value for each different value of TSI.

We observe that \(P_{c\rightarrow e}(x)\) (line-plot 1 in Fig. 10) gets its maximum values half-way through the path from the source to the client, while \(P_{c\rightarrow E}(x)\) (line-plot 2 in Fig. 10) is maximised just after the middle of the delivery path. This was also shown in Figs. 2(a) and 2(b). The grey area in Fig. 10 denotes the area where \(Probcache\) should be maximised, in order to guarantee fair resource allocation between clients. In particular, and as mentioned before, [1], we argue that caching space should be allocated to users in proportion to their distance from the source. Proportional resource allocation translates to maximisation of \(Probcache\) within the grey area in Fig. 10, that is, close to the flow’s destination. In this figure, we observe that when clients are connected one, two or three hops away from the source the value of \(Probcache\) is maximised within the desired grey area, while for clients connected further away, the maximum value of the function falls outside this area.

However, as shown in Fig. 2, even for small values of TSI, where \(Probcache\) takes its maximum value within the grey area, we see that this maximum value is still smaller for these flows than for instances of \(Probcache\) where TSI gets higher values (i.e., for users connected further away from the source node). In turn, this gives constant advantage in terms of caching probabilities to flows connected more hops away from the core of the network. Therefore, we argue that the maximum value of the caching function can not be a measure on its own, but instead it has to be evaluated together with the \textit{density distribution} of the function for each particular instance. The density of the function with regard to the TSI and TSB values of \(Probcache\) is given by its integral, which we calculate next in order to verify our claims.

![Fig. 10. Derivatives of: (1) \(P_{c\rightarrow e}(x)\) (top part of Eq. 10) and (2) \(P_{c\rightarrow E}\) (bottom part of Eq. 10). Grey area shows the ideal interval where maximum values for \(Probcache\).](image)

B. Density Distribution of Basic Function

Integrals capture the density distribution of \(Probcache\) with regard to the values that the function takes for different distances of clients from sources. The distribution of the density relates closely to the cache deployment setting, that is, the caching resources available along the path. For example, in the heterogeneous cache size scenario, where more cache is deployed towards the edges of the network, we expect that the distribution density of \(Probcache\) will increase as we move away from the source of content (i.e., for clients connected far from the source). This is because these clients have more resources available to cache and therefore, \(Probcache\) is more likely to choose these nodes to cache contents of clients travelling longer paths, in order to leave nodes in the core for shorter content flows.

In contrast, in case of equal caches along the path, we expect that all users have equal opportunities to utilise the available resources irrespective of their distance from the source. Hence, in this case the distribution density should be somewhat evenly distributed along the path.

To depict this situation and verify our claims, we initially calculate and then plot the integrals of \(P_{c\rightarrow e}\) (Eq. 7) and of \(P_{c\rightarrow E}\) (Eq. 9).

\[
\int_1^c P_{c\rightarrow e}(x) = \frac{c^2}{6} + \frac{c}{2} - \frac{1}{6c} - \frac{1}{2} \quad (11)
\]

\[
\int_1^c P_{c\rightarrow E}(x) = \frac{1}{2} + \frac{1}{4c} - \frac{3c}{2} + \frac{c^2}{2} + \frac{c^3}{4}
\]

![Fig. 11. Integrals as calculated above (top and bottom part of Eq. 11). In the same Figure, we also sketch the available cache capacity distribution along the path - the grey area - in each of the two cases. The cache capacity distribution is calculated taking into account the fact that clients should cache their contents close to their point of attachment (e.g., in the last two hops of the delivery path) with higher probability (as also shown in Fig. 10). For the homogeneous cache deployment case, for instance, the capacity distribution is the same along the entire path, denoting that all users should have equal chances to cache regardless of their position on the path. Instead, in the heterogeneous cache deployment case, the capacity distribution is increasing as it](image)

The density of the function here is similar to the \textit{expectation} of a probability distribution function of a random variable. However, given that \(Probcache\) may take values higher than 1, which would mean that the path cache resources allow for caching of multiple copies of the chunk in question, \(Probcache\) is not a probability distribution function (and \(x\) not a random variable). Therefore, to find the \textit{density distribution} of \(Probcache\), we calculate its integral.
times into account the extra cache resources available towards the edge of the network. We contend that the distribution of the value of ProbCache should be within this grey area in order to achieve fair content multiplexing in in-network caching environments.

We observe, however, that ProbCache does not achieve distribution density close to the “fair” grey area in any of the settings evaluated. Instead, in both cases, \( P_{c \rightarrow e}(x) \) and \( P_{c \rightarrow E}(x) \) appear more aggressive and with density distributions that favour users connected far from the source. This fact verifies our earlier claim, which we presented in Fig. 2.

Based on our observations, we proceed to deal with this unfair behaviour by modifying the CacheWeight factor of ProbCache in Section IV-B.

X. BEHAVIOUR OF ENHANCED FUNCTION

A. Maximum Values of Enhanced Function

Calculating the derivatives of Eqs. 4 and 5 and minimising the respective functions to find the local maxima gives us the following:

\[
\frac{dP'_{c \rightarrow e}(x)}{dx} = 0 \Rightarrow x = c
\]

\[
\frac{dP'_{c \rightarrow E}(x)}{dx} = 0 \Rightarrow x = \frac{3(1 + c)}{2(2 + c)}
\]

We plot the maximum values of \( P'_{c \rightarrow e}(x) \) and of \( P'_{c \rightarrow E}(x) \), according to Eq. 12 in Fig. 12. We observe that the enhanced versions of ProbCache for both the homogeneous and the heterogeneous cache deployments are behaving in a fairer manner. That is, the functions get their maximum values when contents go through nodes that are close to the content’s destination, as can be seen by line-plots (3) and (4) in Fig. 12 for \( P_{c \rightarrow e}(x) \) and \( P_{c \rightarrow E}(x) \), respectively - note that in Fig. 12 line plot (3) is overwritten by the upper borderline of the grey area. Clearly, this is a fairer behaviour than the one adopted by line-plots (1) and (2) in the above figure, which depict the performance of the original algorithm, as we also show through extensive simulations in Section V.

B. Density Distribution of Enhanced Functions

As a final step, we calculate and plot the integrals of the enhanced versions of the algorithm. Our intention, as before, is to monitor the density distribution of the enhanced algorithm and see whether it falls within the grey area presented in Fig. 11, which we consider as the ideal behaviour in terms of multiplexing between competing content flows along a path of caches. Our calculations give the following:

\[
\int_{1}^{c} P'_{c \rightarrow e}(x) = \frac{2c - \left(\frac{1}{2}c\right)^{2}(1 + c)}{2 + c}
\]

\[
\int_{1}^{c} P'_{c \rightarrow E}(x) = -\frac{c(12 - c(5 + 4c)) + \left(\frac{1}{2}c\right)^{2}(-5 + c(-2 + c(3 + c)))}{(2 + c)(3 + c)}
\]

We plot the integrals of the enhanced function, namely the top and bottom parts of Eq. 13 in Fig. 13.

Indeed, we see in Fig. 13 that the density distribution of both versions of the enhanced ProbCache fall within the grey area. We argue that this is the ideal behaviour in terms of resource management and allocation in in-network caching environments. In Fig. 13(a) and for line-plot (2), which represents the enhanced version of ProbCache, we see that all users have equal overall chances of caching their

Fig. 12. Plots of: (1) \( P_{c \rightarrow e}(x) \) (top part of Eq. 10), (2) \( P_{c \rightarrow E}(x) \) (bottom part of Eq. 10), (3) \( P'_{c \rightarrow e}(x) \) (top part of Eq. 12), (4) \( P'_{c \rightarrow E}(x) \) (bottom part of Eq. 12). In contrast to the original algorithm, the enhanced version gets its maximum values within the desired area, where fair content multiplexing is guaranteed, according to our design principles. This is the case for line-plots (3) and (4) in the above figure - note that line-plot (3) is overwritten by the upper bound of the grey area.
contents over the entire path, regardless of their distance from the source. Clearly, there is significant improvement compared to line-plot (1) in Fig. 13(a), where the further away users are (in terms of hops from the source), the higher the chances they have to cache their contents at some point along the delivery path. The situation is similar in case of heterogeneous cache sizes as shown in Fig. 13(b).

XI. SCENARIO 3: HETEROGENEOUS CACHE ENVIRONMENT

We repeat the same experiment as in Scenario 1 (Section V-B), but this time we deploy heterogeneous amounts of caches to routers along the path. In particular, as we move away from the server we increase the amount of traffic that can be temporarily stored in the routers’ cache memory. We associate the cache capacity of each router with its distance from the server(s), according to the formula $N_i \cdot TSB$ (see also discussion in Section VIII). Although this might be a slightly unrealistic setting for real deployments, due to the large number of servers or server farms deployed in relatively arbitrary locations, it gives us a good insight of the protocols’ behaviour in heterogeneous cache size environments. We consider that the analysis of topological features (including cache sizing) is a subject of a different study, which we have initiated in [4]. We report that the algorithm adjusts its behaviour also in case of random cache sizes throughout the network, as assumed earlier in Section III-C. We do not present those results in this paper due to space limitations.

The results of the heterogeneous cache size environment are presented in Fig. 14. As we have also shown in [1], ProbCache becomes aware and exploits the extra cache resources along the path. The performance difference of $ProbCache^+$ in terms of Server Hits is now wider than before and reaches to almost 13% compared to LCD, LCEd and $CE^2$ (see Fig. 14(a)). Furthermore and in contrast to the previous experiment, in the heterogeneous cache size setup, $ProbCache$ reduces significantly the Hop Reduction Ratio, as can be seen in Fig. 14(b). The difference of $ProbCache^+$ is in the order of 7%, compared to LCD, LCEd and $CE^2$, which is rather significant, if we consider the topology characteristics (i.e., mean valence of the degree in the topology is 2), hence, very few paths are longer than 6-7 hops.

Finally, in terms of content multiplexing fairness, we observe in Fig. 14(c) that the fairness performance of the original version of ProbCache has now increased. This is a somewhat unexpected result, if we consider that the amount of cache increases as we move towards the middle and the edge of a given path, where ProbCache gets its highest value and tends to cache contents (see also Fig. 2(b) in Section IV). On the other hand, ProbCache+ behaves as expected; it pushes contents towards the edges of the paths, while at the same time it manages resources in an efficient way and results in more contents being cached along the path. As we have shown in our second evaluation scenario (see Section V-C, although probabilistic caching might not cache potential popular content immediately, the design of ProbCache converges to the identification (and therefore, caching) of popular content almost as fast as $CE^2$.

We have evaluated the performance of five different caching protocols with regard to their resource management and allocation properties. Our evaluations included a variety of different network settings, traffic conditions and patterns, as well as various performance metrics. We conclude that deterministically caching content chunks in fixed and pre-determined places along a delivery path results in poor resource management and in turn, reduced overall network performance. In contrast, sophisticated algorithms that take into account the amount of available cache resources along the delivery path improve both network utilisation and user-perceived quality of service. This was demonstrated by significant reduction in server hits, which in turn, reduces the delivery time; huge reduction in the number of cache evictions (with parallel increase of cache hits) in case of ProbCache; and fairer behaviour according to the CMFI index introduced here.

Finally, caching at carefully selected places in the network does not degrade the convergence performance of caching protocols in case of small scale flash crowd events, as we demonstrated in Section V-C. In contrast, we have shown that ProbCache performs almost similar and sometimes even better than universal caching, which is clearly the best candidate when it comes to flash crowd events. In case of ProbCache this is due to sizing content caches according to the amount of traffic that they serve per unit time, which results in identification of popular contents much faster than we initially expected. As noted before, this is the case for a wide range of Zipf parameters, as can also be seen in [1], where we used $a = 0.8$ and our results here, where $a = 1.2$. 

![Server Hits](image1)

(a) Server Hits

![Hops Reduction](image2)

(b) Hops Reduction

![Content Multiplexing Fairness Index (CMFI)](image3)

(c) Content Multiplexing Fairness Index (CMFI, see Eq. 6) for Heterogeneous Cache Environments

Fig. 14. Scenario 3: Heterogeneous Cache Environment - Overall Results, Set of Simulations Applying Increasing Amount of Cache in each Experiment. Cache size is set according to the formula $N_i \cdot TSB$, where TSB is the Time Since Birth value.

A. Summary of Results

![Graph](image4)
We note that our request patterns do not incorporate temporal or spatial locality characteristics. We expect that correlations between temporal locality and cache size and spatial locality and line size (i.e., path length) will improve the performance of ProbCache. In particular, good temporal locality with larger caches will increase cache hits and good spatial locality (which can be incorporated in the TSI and TSB values) will reduce cache misses when the path length increases. These directions comprise topics of our future work, where our evaluations will be based on real Internet traces.

XII. IN-NETWORK CACHING STATE OF THE ART

In-network content caching has attracted a lot of attention recently, mainly because of its inherent capability to enhance content distribution in the Internet. Although a part of the research community argues that in-network caching and traditional content distribution (in terms of overlays and CDNs) have conflicting goals [5], [6], it is our position that these two research fields are complementary. Content distribution optimisation has been the subject of several recent studies, e.g., [7], [8], [9], [10]. In our opinion, such studies can serve as input to the design and implementation of in-network caching approaches. Although the specific techniques that will bridge the gap between these two approaches to content distribution are not yet identified, there have been several attempts for individual optimisation from both sides. Here, we focus mainly on recent advances and research findings in the field of in-network caching as well as on modelling and implementation studies in the area.

One group of studies focuses on the feasibility and potential in terms of performance gains of in-network caching as a technology to optimise content delivery (e.g., [11], [12], [13], [14]). For example, studies such as [15] and [16] have provided analytical models to assess the gain of in-network caching in terms of network utilisation, while [17] and [18] have elaborated on performance and cost feasibility issues for cache-enabled Internet routers from a hardware implementation perspective; they found that indeed, today’s technology can support in-network caching, although the growth of Internet traffic will challenge some of the design principles.

In [12], the authors provide a comprehensive performance evaluation of in-network caching taking into account several parameters, such as content request distributions, the catalog size, and cache replacement policies. They conclude that content popularity is (by far) the most important parameter. In [19], the authors apply load-balancing techniques depending on the popularity of content objects and find that multi-path routing can increase the overall network performance. Along the same lines, authors in [6] 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 [20], the authors report savings of up to 20% in terms of hops that requests need to travel before they hit cached contents, based on a combination of traces from BitTorrent and CAIDA. In [21], the authors are concerned with redundancy elimination. They design an efficient algorithm for realising and reducing duplicate traffic at the network level and report significant bandwidth savings. Traffic redundancy elimination is complementary to caching redundancy, which is the subject of the present study.

In [22], 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 in [4] and in [1], by caching a limited number of copies of a chunk in selected caches in the network, we achieve significantly higher gains. In particular, in [4], 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 by up to 15% in scale-free topologies.

The second group of studies, which is closer to our work in this paper, have targeted the topic of caching policies to improve overall network performance. In [23], the authors propose a combined scheme for potential based routing and a random caching policy that is shown to improve delivery delay, although the paths chosen to route contents may not be the shortest ones. In [24], the authors propose a collaborative caching and forwarding design, which makes caching decisions based on content popularity. In [25], the authors introduce a technique, similar in nature to LCD [3], according to which the number of chunks of a content file that get cached in the network increases with the number of hits that the file gets. Finally, [5], proposes advertisement of cached contents in the control plane to reduce the uncertainty of a totally uncoordinated in-network caching environment. Although, as expected, scalability is the main challenge of such an approach (as we have also discussed before), the authors show that this approach deserves further attention. It is questionable, however, whether the collaboration between the network and the control plane can take place at the chunk-level. Finally, in [26], we have recently proposed off-path caching techniques complemented hash-routing of requests to cached contents. We have seen that in many cases co-ordinated off-path caching outperforms on-path caching techniques (such as ProbCache), but careful design is needed in order to avoid extensive delays, as off-path caching inherently requires longer paths. We have made the first steps towards this direction in [26].

Although the above studies are relatively close to our investigations in this paper, none of them is considering in-network resource management as a central challenge. The solutions proposed above are therefore targeting a different design space and are considered complementary to our caching approach proposed here. ProbCache can be implemented as a resource management and content flow multiplexing technique and work in tandem with most of the above studies.

REFERENCES


