Efficient Hash-routing and Domain Clustering Techniques for Information-Centric Networks[☆,☆☆]

Vasilis Sourlas[a], Ioannis Psaras[a], Lorenzo Saino[a], George Pavlou[a]

[a] *Department of Electronic and Electrical Engineering, University College London, UK.*

Abstract

Hash-routing is a well-known technique used in server-cluster environments to direct content requests to the responsible servers hosting the requested content. In this work, we look at hash-routing from a different angle and apply the technique to Information-Centric Networking (ICN) environments, where in-network content caches serve as temporary storage for content. In particular, edge-domain routers re-direct requests to in-network caches, more often than not off the shortest path, according to the hash-assignment function. Although the benefits of this off-path in-network caching scheme are significant (*e.g.,* high cache hit rate with minimal co-ordination overhead), the basic scheme comes with disadvantages. That is, in case of very large domains the off-path detour of requests might increase latency to prohibitive levels. In order to deal with extensive detour delays, we investigate nodal/domain clustering techniques, according to which large domains are split in clusters, which in turn apply hash-routing in the subset of nodes of each cluster. We model and evaluate the behaviour of nodal clustering and report significant improvement in delivery latency, which comes at the cost of a slight decrease in cache hit rates (*i.e.,* up to 50% improvement in delivery latency for less than 10% decrease in cache hit rate compared to the original hash-routing scheme applied in the whole domain).

Keywords: information-centric networks, cache aware routing, off-path in-network caching, nodal clustering/partitioning, hash routing.

1. Introduction

Internet usage patterns have been constantly changing over the last decades, reaching a situation that was not foreseen when it was originally designed. The

[☆]The present manuscript is published in *Computer Networks*, Volume 103, July 2016, pages 67-83. The published article is available at `http://dx.doi.org/10.1016/j.comnet.2016.04.001`.

Email addresses: `v.sourlas@ucl.ac.uk` (Vasilis Sourlas), `i.psaras@ucl.ac.uk` (Ioannis Psaras), `l.saino@ucl.ac.uk` (Lorenzo Saino), `g.pavlou@ucl.ac.uk` (George Pavlou)

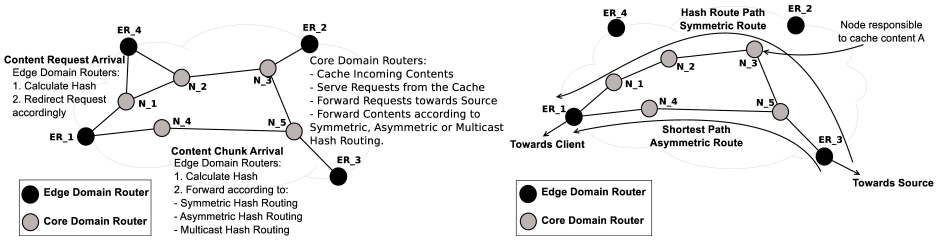
engineering principles underpinning today's Internet architecture were created in the 1960s and 1970s with the assumption that Internet would be mainly used for host-to-host communications. Instead, nowadays, the Internet is increasingly being used for content dissemination and retrieval and this trend is forecast to continue in the foreseeable future [1].

This mismatch between the original design assumptions and current usage patterns has partially been addressed through application layer solutions such as Content Delivery Networks (CDN) and Peer-to-peer (P2P) overlays, which have retrofitted some desirable content-aware functionalities on top of the existing architecture. However, the lack of native network support for content distribution restricts the efficiency of such approaches, and also potentially hinders the evolution of the Internet as a whole.

This has created a trend towards content-oriented networking, which has recently been realised through the Information-Centric Networking (ICN) paradigm. Information-Centric Networking, similarly to P2P and CDNs, puts content itself in the forefront of attention when it comes to content delivery. That is, content can be delivered from any network location/device, provided that this device holds a valid copy of the requested content.

Extending the P2P paradigm, where mainly end-user devices can serve requests for content, the ICN paradigm also includes in-network devices, that is router caches, as potential content servers. Based on this principle, a new field of research has emerged coined "in-network caching". The challenges of addressing content temporarily stored in router-caches (from now on referred to as routers, or caches), resolving the location of this cache and fetching the content from the corresponding network location has recently attracted considerable attention [2]. Such challenges include caching redundancy, efficiency in utilising in-network storage [3] and replacement of content in caches according to their popularity [4], to name a few. Last but not least, an ICN network operates based on packet-sized *content chunks*, instead of content objects (or files), as is the norm in case of overlay/proxy caching. This fact adds one more requirement to the operation of in-network content caches - that of *line speed operation*.

In this paper, we deal with the resolution of requests to in-network content caches in domain-wide environments and the optimisation of the resolution process in order to increase cache hits, but also keep the latency needed to reach the cache under certain limits. Our cache-aware routing scheme utilises hash-routing techniques, which have been proposed in the past for mapping requests to physically co-located servers [5], [6]. According to hash-routing, each element within the network (be it servers in server-racks, or routers within a domain network) are assigned with a range of the hash space and store the content items whose hashed identifiers fall within the node's hash space. In contrast to alternative architectural proposals according to which extra resolution steps are essential (*e.g.*, [7], [8], [9]), this operation avoids complex request-to-cache resolution and minimises signalling overhead (the only overhead is the calculation of the hash function itself).



(a) Hash-routing Functional Architecture

(b) Symmetric and Asymmetric Hash-routing

Figure 1: Hash-routing Functional Architecture and Symmetric/Asymmetric options for Request and Content Routing.

1.1. Background on Hash-Routing

Similar to the work in [10], we also target here domain-wide ICN deployments, where a content naming scheme, flat or hierarchical, is in place. Also, the edge-domain routers implement a hash function that determines both the content placement and the request-to-cache routing process. In particular, when an edge router receives a content request, calculates the hash of the content identifier and redirects it to the responsible cache. If the requested content is cached in the corresponding router, the content is returned to the client, otherwise, the request is forwarded towards the original server. In a similar way, incoming content items responded for the origin server are forwarded for caching (or not) according to the hash of their identifier. As it is described in [10], the main concept underpinning our approach is that *a content can be opportunistically found in a domain only in the cache calculated by the hash function.*

The hash-routing schemes proposed in this paper require edge-domain routers and cache nodes to implement a hash function which maps content identifiers to cache nodes. This function is used: *i)* by cache nodes to identify the set/range of content identifiers or names that they are responsible for, and *ii)* by edge routers to route requests to the corresponding cache node (see Figure 1(a)). As a result of this approach, each content object can be cached in a domain at most once, thus preventing redundant replication of cached content and resulting in more efficient utilisation of cache space. This approach also allows edge routers to forward content requests to the designated cache directly, without performing any lookup. In addition, the intra-domain forwarding procedure is performed without requiring any sort of inter-cache co-ordination since the hash function can be computed in a distributed manner by edge routers and caches, thus being scalable to any domain size.

The hash function maps a content identifier (flat or hierarchical) to a caching node of the domain. Such function does not need to produce a cryptographic hash. In fact, it is desirable for its output to be produced with minimal processing as long as it is capable of mapping content items to cache nodes so that the load of caches is evenly spread.

For example, in case of human-readable identifiers, like URLs (RFC 3986, [11]), content items can be mapped to caching nodes by hashing their identi-

fiers using fast non-cryptographic hashing functions such as Murmur, Jenkins, xxHash, CityHash and CRC32 and then applying modulo hashing over the number of caching nodes. In case of binary content identifiers, such as those defined by RFC 6920 [12], modulo hashing can be applied directly on content names.

Consistent hashing [6] may also be used to minimize the number of items to be remapped as a result of failures or additions or removals of caching nodes. The choice of the specific hash function is out of the scope of this paper, since it does not affect the optimisation process proposed here, and each network manager can choose the one that fits its own specifications.

In [10], we have designed and evaluated the performance of five different algorithms that take advantage of hash-routing techniques in ICN environments. The differences between the five algorithms proposed in [10] lie in the routing and replication process followed for requests and content objects, respectively. In particular, it is clear from Figure 1(b) that hash-routing techniques can follow symmetric or asymmetric paths to deliver the content. In the interests of space, we omit the details of the different hash-routing techniques investigated in [10] and refer the reader to that paper for further details.

1.2. Contributions

Hash-routing falls in the group of *off-path* in-network caching techniques, as opposed to *on-path* in-network caching (*e.g.*, [3], [13] - see further discussion in Section 2), according to which content is fetched from caches, only if the request "hits" the content along the path to the content origin. Although on-path in-network caching techniques by definition do not require any co-ordination between cache nodes, they result in suboptimal performance, as we have already demonstrated in [10]. In contrast, off-path caching techniques improve performance in terms of cache hits, but come at the cost of extra coordination, *e.g.,* [7], [14].

Hash-routing techniques clearly improve the performance of the cache network in terms of cache hits, as shown extensively in [10] and at the same time require minimal co-ordination among network nodes. However, this increase in cache-hits comes at the cost of increased latency caused by the detouring required to look up the responsible cache. This tradeoff is clearly affected by the number of extra hops to be travelled off-path to find the cached content. In case of a small network/domain, this latency might be negligible, but as the size of the network increases (and depending also on topological characteristics, such as the density of the network graph, its diameter, *etc.*), the increase in latency might become prohibitive.

Building on this tradeoff, in this paper, we investigate the potential of partitioning the network using node clustering techniques to keep the latency under certain bounds. We introduce the system model in Section 3. We make use of the well known "k-split clustering" [15], [16] and "k-medoids clustering" [17] techniques and we also introduce a new "Bin packing content assignment function" (Section 4). Our results show that by splitting large domains in clusters the latency to hit and deliver cached content indeed drops. Inevitably and expectedly, this comes at the cost of lower cache hit rates (up to 50% improvement

in delivery latency for less than 10% decrease in cache hit rate). The choice of whether an ISP makes use of hash-route clustering and which clustering technique to use depends on the interests and the business model of the particular ISP. We finally lay down the options different ISPs might have according to domain sizes.

2. Related work

The topic of in-network content caching has received wide attention recently in the context of Information-Centric Networks. Generally speaking, and according to [2], the in-network caching problem can be split in three distinct subproblems, which we next discuss in turn. These are the allocation of caches to network routers and the economic incentives of ISPs and other market players to adopt the new networking paradigm; the placement of content into the caches and the subsequent discovery and retrieval of content from the network caches.

2.1. Cache Allocation and Economic Incentives

First and foremost comes the issue of allocating cache space in network routers. This includes both the capital expenditure needed in order to place cache memory in routers, but also the decisions as to where should memory be allocated inside the network.

The economic implications of in-network caching have been investigated in recent studies, focusing on the incentives needed by ISPs and content providers to deploy and operate in-network caches [18], [19], [20]. Furthermore, inter-ISP issues related to in-network caching (*e.g.,* savings from transit traffic) and new pricing models have been investigated in [21], [22], [20]. Motivated by similar concerns on the cost implications of caching, the authors in [23] design a "Cost-Aware cache decision policy", based on which caches hold the content that will bring the most savings to the ISP in terms of transit traffic.

2.2. Content Placement

The content placement problem deals with the placement of content in caches. The placement can be either *proactive* or *reactive*. In *proactive content placement*, the operator decides on which content should be cached where in an offline manner [24], [25], [26]. *Proactive caching* has been extensively used in CDN networks and overlay cache networks. In Information-Centric Networks, instead, *reactive content placement* is the dominating strategy. Among other reasons, this is because caching and data retrieval operations happen at line-speed. This is in contrast to CDN settings, where the CDN operator can place/cache and migrate content proactively according to expected demand. As such, most works in this area have focused on placing content in in-network caches in order to optimise traditional caching metrics such as *delivery latency* based on content popularity assessment [4], [27], content locality [28], or cache redundancy and cache resource management [3], [13].

2.3. Request to Cache Routing

After proactively or reactively placing content in caches, the system needs to poses the right mechanisms in order to direct content requests to the right cache. Request to cache routing can follow one of two approaches, either *oppostunistic on-path*, where content is searched on-path as the request is travelling towards the content source, or *co-ordinated off-path*, where requests are forwarded off the shortest path to some designated cache that is likely to hold this content. The most prominent solution to this problem is to hold an extra routing table which maches requests to content items cached in nearby nodes. Representative proposals in this space are [29], [30], [31] and [32].

The techniques proposed in [7], [8] and [14] use co-ordination techniques between the data and the control plane to place content and re-direct requests to the corresponding caches. Finally, in [33] two methods are proposed to route requests to the nearest replica of a content by either flooding requests or meta-requests to discover the content location. We argue that such approaches introduce considerable amounts of overhead and delay and as such result in inherently less scalable solutions. Last but not least, recently, the authors in [34] have proposed scoped flooding-based content discovery. The proposal includes a ring model, which limits the spread of the flood to the neighbourhood. The results show that although there is some overhead, the scoped-flooding approach is far from prohibitive and can in fact scale and achieve considerable gains.

2.4. Balancing Tradeoffs

Both on-path and off-path caching present trade-offs. On-path content caching requires less co-ordination and management, but may provide limited gains. Conversely, off-path content placement and retrieval can attain higher hit rates at the cost of extra co-ordination and communication overhead. Co-ordination overhead refers to the decision making process of where to cache incoming content, as well as to the forwarding rules that (re-)direct incoming requests to cached content [35], [36]. Communication overhead, on the other hand, refers to the redirection of requests and the retrieval of content from off-path caches, which involve possibly traversing longer paths and therefore, increasing network load [7], [35], [36], [37].

Hash-routing techniques are off-path caching techniques, but the adoption of a hash function to determine the location of cached content eliminates the need for inter-cache co-ordination typical of other off-path caching techniques. Hash-routing is a widely used technique in the context of Web caches [5], [6], but only to route requests among co-located nodes.

In [10], we showed that hash-routing techniques can be successfully used in a domain-wide ICN environment to maximise cache hits without any co-ordination requirement. Subsequent to our work, similar techniques have been proposed. CoRC [9], for example, proposes to route content requests based on their hash, but with the specific purpose of reducing the size of routing tables rather than improving caching efficiency.

In this work, we further extend the state of the art in hash-routing techniques by providing a practical way to bound latency, hence making it suitable

for topologies of arbitrary size. In particular, we argue that in case of large network domains, the hash-route-based re-direction of requests results in very long paths, hence, prohibitively increasing the latency to reach cached content. We therefore, introduce domain clustering techniques, which are applied in an offline manner and split the domain in smaller clusters. In turn, each cluster is in charge of a separate hash-function which only applies to the specific nodes in that cluster. We argue and show that domain clustering reduces excessive latency, due to hash-routing, but at the same time achieves considerable cache hit rate performance.

3. System Model and Problem Formulation

We consider an information-centric network of arbitrary topology, which can be represented as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Let $\mathcal{V}$ denote the set of ICN routers and $\mathcal{E}$ the set of communication links connecting them. Let also C_v be the storage capacity (in bits) of router $v \in \mathcal{V}$. Throughout the paper we will use the calligraphic letters to denote sets and the corresponding capitals for cardinality; for example $|\mathcal{V}| = V$.

Let $\mathcal{M}$ denote a given and fixed set of M content items that have to be delivered over the network. Throughout this paper, we assume that all items are of equal size and all nodes have the same storage capacity ($C_v = C$, $\forall v \in \mathcal{V}$)[1]. We normalise the size of each item to one unit with respect to node's storage capacity ($s^m = s = 1$, $\forall m \in \mathcal{M}$) and, hence, each node can hold up to C different unit-sized items. Note that the NDN [38] information-centric architecture segments content items into smaller, individually-named, pieces to allow flexible distribution and receiver-driven flow control.

Upon receipt of a request, an edge router calculates the hash of the content identifier and forwards it to the responsible cache. In the case of cache hit, the content is returned to the client, otherwise the request is forwarded towards the original server. We denote by $\boldsymbol{o} = \{o^1, \ldots, o^M\}$ the set of origin servers for every content item, where $o^m \notin \mathcal{V}$ the origin server regarding information item $m \in \mathcal{M}$ (typically a node outside the examined domain). We also denote with $\boldsymbol{e} = \{e^1, \ldots, e^M\}$ the egress nodes for every content item, where $e^m \in \mathcal{V}$ is the domain node through which a request will be forwarded towards the item's origin server o^m.

Content requests are generated with rate $\boldsymbol{r}_v = \{r_v^1, \ldots, r_v^M\}$, where r_v^m denotes the aggregate incoming request rate (in requests per second) generated by users attached at node/cache v for item m. Vector $\boldsymbol{r}_v$ is an estimate of the actual request pattern based on observed, historical content access data (within a given time window). This estimate is used as a prediction for the future number of requests addressed to each node. The optimal way to perform this estimation is out of the scope of this paper, but we adopt an approach similar

[1] In the evaluation section we also consider a scenario with different storage capacities among the nodes of the network.

to [39], using an exponential moving average function in each measurement window.

We approximate item popularity by a Zipf law distribution of exponent z [40], [41]. Generally, the popularity of each content item may differ from one geographic location to another, a phenomenon that is referred to as locality of interest (*spatial skew* in [42]). In our model, this is captured through a localised request generation model, where the aggregate request pattern $\boldsymbol{r}_v$ is different across regions. We assume V different regions each served by an ICN router. All regions are characterised by the same value for the Zipf distribution exponent which captures the global popularity of items and at each region the ranking/order of the items within the Zipf distribution is different, which captures the different locality of interests.

A given hash assignment function ρ maps a content item identifier to a cache/node in the domain. Let binary matrix $\boldsymbol{Y}(\rho) \in \{0,1\}^{M \times V}$ denote hash mapping $y_v^m(\rho) \in \{0,1\}$ on whether content item m is mapped to node v according to the hash assignment function ρ. Ideally, the used hash function should be capable of mapping content items to caches so that the load of caches is evenly spread and the items are cached as close as possible to the requesting users. In order to increase the utilisation of the intra-domain cache space, we assume that each content item can be cached in a domain at most once, thus preventing redundant replication of the items and increasing the cache hit within the domain. So in this paper we assume that $\sum_{v \in \mathcal{V}} y_v^m(\rho) = 1, \forall m \in \mathcal{M}$. The same equation also holds in the partitioned domain (see Section 4), wherein this case an item is cached at each cluster/sub-domain at most once.

We denote by $p_v^m(t) \in \{0,1\}$ the probability of finding information item m cached in the domain's node v at time t, where $y_v^m = 1$. This probability is a function of the dynamics of the content item's popularity assigned to the corresponding cache node, the dynamics of the popularity of all the items assigned to the same node, as well as the cache size of the corresponding node. In other words:

$$p_v^m(t) = f(\hat{r}_v^m, \boldsymbol{r'}, \boldsymbol{Y}(\rho), \mathcal{P}_v(\rho), C) \tag{1}$$

where: (1a)

$$\hat{r}_v^m = \sum_{j \in \mathcal{V}} r_j^m, \tag{1b}$$

$$\boldsymbol{r'} = \{\sum_{j \in \mathcal{V}} y_j^1 r_j^1, \ldots, \sum_{j \in \mathcal{V}} y_j^M r_j^M\}, \tag{1c}$$

$$P_v(\rho) = \sum_{i=1}^{M} y_v^i. \tag{1d}$$

Note that $\mathcal{P}_v(\rho)$ is the set of items that the hash assignment function ρ has assigned to node/cache v, and $P_v(\rho)$ is its cardinality.

When the capacity of a cache node v is larger than the total size of the content items that the utilised hash function has assigned to it ($C \geqslant P_v(\rho)$),

then the $p_v^m(t)$ probability is always equal to one. In a real scenario, however, the total cache capacity of the domain is usually smaller than the total size of the information space and as such the hash function assigns to a cache more items than those it can permanently store. Generally, the amount of items that are assigned by a hash function at each cache is a function of the total capacity of the domain. For example, a modulo-like hash function assigns $P_v \approx C_v \frac{M}{\sum_{v \in \mathcal{V}} C_v} = \frac{M}{V}$ items at each cache node $v \in \mathcal{V}$, assuming equal cache capacity among the network nodes. In those cases the probability of finding an item at a cache is a time varying function that depends on various network dynamics as shown in Eq.(1). The evolution of an information item cached in a node assuming an LRU replacement strategy can be modelled as an absorbing Markov chain; in [43] we provide a thorough analysis of the time required for an item cached at a node to be discarded. On the other hand, an exact formula for probability $p_v^m(t)$ cannot be exported by the analysis of such a Markov chain, since this relies not only on the dynamics of the items assigned to a cache but also on the evolution of an item within the cache itself (*e.g.*, in which position is the item located and which items are in front or behind it in the cache) [44].

A request for content item m generated at node u at time t, incurs a cost equal to $D_{vu}(t)$, if served by node v ($y_v^m = 1$). Parameter $D_{vu}(t)$ captures the cost (*e.g.,* latency, monetary cost) of transferring content from node v to node u when the request is issued from a user attached at node u at time t. In this work, we consider the retrieval latency as one of our basic metrics and we have:

$$D_{vu}(t) = \begin{cases} d_{vu} & \text{if } p_v^m(t) = 1, \\ d_{e^m u} + d_{o^m e^m} & \text{if } p_v^m(t) = 0, \end{cases} \tag{2}$$

or else

$$D_{vu}(t) = d_{vu} \cdot p_v^m(t) + (d_{e^m u} + d_{o^m e^m}) \cdot (1 - p_v^m(t)), \tag{3}$$

where d_{vu} is the actual latency (in *ms*) between nodes v and u. When an item is not cached within the domain it is fetched from the corresponding origin node through the corresponding egress node.

From Eq.(3) it is obvious that the incurred latency is also a function of the capability of a domain to cache content, the efficiency of the used hash function (how far is the requested item assigned from the area of interest) and the size of the domain (the larger the domain, the larger the distance between its nodes).

The main contribution of this paper is based on this last observation. *Since the distance that a request has to travel before it can reach the node responsible for caching the hashed identifier depends on the size of the domain, we address this issue by splitting large domains in smaller hash-routing clusters.* In the following, we discuss the algorithms used for domain clustering, as well as the specifics of the hash-assignment functions used in this study.

Discussion

Deriving an exact, closed-form equation for $p_v^m(t)$ is subject of a separate study and hence, out of the scope of this paper. Therefore, here, in order to

better understand the correlation between: *i)* the storage capacity of a node, *ii)* the population of items assigned at each cache, and *iii)* the probability of finding an item at a cache, we use the analysis presented in [45] (also known as Che's approximation), for obtaining approximate hit/caching probabilities of the LRU replacement algorithm. Che's approximation introduces the concept of *characteristic time* ct_m of each item m in a cache, which corresponds to the time spent by an item in a cache from insertion to eviction. In other words is the time needed for the $P_v - 1$ remaining items stored in a cache (above m) to be requested C_v times and thus evict item m from the cache of node v, given that item m is not re-requested within this interval. The characteristic time is a random variable specific to each item, but in [45] a mean field approximation is used to derive a single constant characteristic time for all items of a cache ($ct_m = ct, \forall m \in P_v$). This approximation has been shown to be very accurate under various conditions [46].

The characteristic time ct of a cache v of size C_v items subject to a stationary demand of the items $\Pi_v = \left\{\pi_v^1, \pi_v^2, \ldots, \pi_v^m, \ldots, \pi_v^{P_v}\right\}$ is the value ct for which:

$$\sum_{m=1}^{P_v} 1 - e^{-\pi_v^m ct} = C_v \Rightarrow \sum_{m=1}^{P_v} e^{-\pi_v^m ct} = P_v - C_v, \tag{4}$$

where π_v^m is the request probability of item m.

The characteristic time can then be used to compute the average cache hit probability for each item. For example, the cache hit probability of item m at cache v assuming an LRU cache replacement strategy is:

$$\bar{p}_v^m = 1 - e^{-\pi_v^m ct}. \tag{5}$$

The above approximation is for LRU caches, but in [47] authors demonstrated its applicability to a large variety of cache replacement policies including FIFO, Random and k-LRU.

Although Che's approximation is a tractable and remarkably accurate approximation, it does not allow to express cache hit probability in a closed form. This is because Eq. 4 cannot be formulated to as a function of the characteristic time, which instead needs to be computed numerically. A closed-form formula for the computation of the average cache hit probability is derived in [48], although it is valid only under the assumption that content popularity is Zipf-distributed. The closed form formula is the solution to a K-order polynomial equation, which can be solved in arbitrary accuracy by increasing K. Due to space limitations, we refer the interested reader to [48] for the exact formula when $K = 2$ and $K = 3$. Instead, here, we present in Figure 2 the hit probability of each item for a given storage capacity per node and for different number of items P_v assigned at each node. We present the hit probability for two skewness parameters, $z = 0.1, z = 0.7$ for the input Zipf-distributed demand.

From Figure 2, as well as from the analysis presented above we observe that as we increase the number of items that are assigned for caching at each node (due to limited number of nodes to apply the hash function) the corresponding

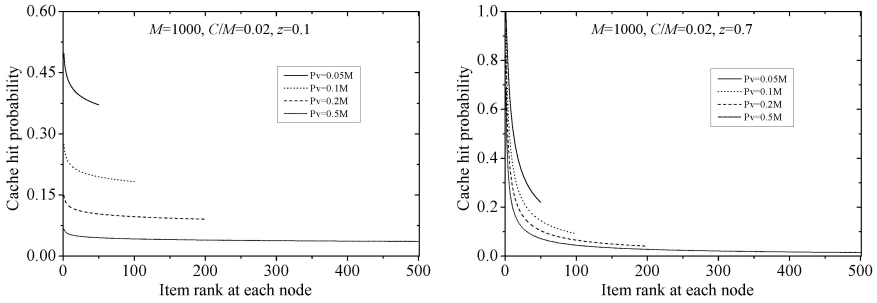


Figure 2: Per item average cache hit probability for various number of items assigned at a node of given storage capacity and for two skewness parameters for the input Zipf-distributed demand.

cache hit probability decreases significantly. According to Eq. 3, this leads to increased retrieval latency. Despite the fact that the analysis in [48] requires a power law demand for the set of items that are assigned at each node and, hence, cannot be directly applied here, it is a useful mechanism to better comprehend the trade-off between the cache hit ratio and the incurred retrieval latency. This will become even more clear in the performance analysis later on. Note here that due to the strict requirement for power law demand distribution we cannot use the analysis in [48] as an alternative and further provide the possible performance gaps to be obtained between the results from a general case and that from the Zipf-distributed demand. The closed form formula in [48] can only be used in the extreme case where we partition a network of V nodes in V clusters. In that extreme scenario all the items are assigned at each node and the power law demand distribution of the requests at each node is the same with the demand at the corresponding cache.

4. Nodal Partitioning and Hash routing

From the above analysis it is evident that *there exists a trade-off between the cache hit ratio* (probability of finding a requested item cached within the domain *i.e.,* Eq. (1)) *and the incurred latency* for the retrieval of an item (*i.e.,* Eq. (3)). In particular, for a given hash function ρ and a given total amount of cache deployed in some domain, the larger the domain, the larger the overall cache capacity (assuming equal cache capacity among the network nodes), which means less items assigned to each node ($\mathcal{P}_v(\rho), \forall v \in \mathcal{V}$) something that in turn increases the overall cache hit ratio (*i.e.,* $p_v^m, \forall v \in \mathcal{V}$ and $\forall m \in \mathcal{M}$). On the other hand, larger domains implies larger intra-domain distances/latencies between the node issuing a request and the node where the item is assigned.

In this section we describe a method to significantly decrease the retrieval latency of a content item at the expense of a tolerable decrease in the cache hit ratio for a given hash function. We also introduce a new offline centralized content assignment function (in Section 4.2) that further reduces the content retrieval latency and enhances the load balancing capabilities of the underlying

routing scheme at the cost of increased cache co-ordination among nodes, retaining at the same time similar cache hit ratio performance with other more distributed hash functions (*e.g.,* random or modulo hash functions).

4.1. Domain clustering/partitioning techniques

We make use of algorithms originally proposed in the area of data mining to leverage the above mentioned trade-off. Specifically, partitioning/clustering of the domain into clusters/sub-domains based on certain similarity metrics can both significantly reduce the average retrieval latency of an item and retain a large cache hit ratio.

Partitioning a domain of V nodes into N clusters facilitates hash routing at cluster-level. Thus, it can substantially reduce the size of the sub-domains decreasing in that way the distance between the requesting and the serving nodes. On the other hand, the application of a hash function at a smaller set of nodes means that a larger set of items is assigned at each node, something that might decrease the cache hit ratio of the system, deteriorating the efficiency of the hash routing technique.

4.1.1. k-split domain clustering

The problem of clustering a set of nodes in a set of $\mathcal{N}$ clusters is generally NP-hard [49]. Here, we use a well defined low-complexity clustering scheme from the literature as one of our partitioning techniques. Particularly, we adopt the k-split clustering algorithm proposed in [15] and [16] in order to partition the initial domain. The k-split algorithm clusters the domain into k clusters and its objective is to minimise the maximum inter-cluster distance based on a similarity metric. This means that initially a representative similarity index I (metric) has to be derived for the nodes of the domain in order to partition the nodes in sub-domains. In this paper we use two different similarity metrics alongside with the k-split algorithm.

Similarity metrics

In its simplest form, the actual topological latency/distance between two domain nodes can be considered as a single dimensional feature for clustering. In this case, similarity of two nodes v_1 and v_2 is captured by their topological distance (*e.g.,* hop count, latency, *etc.*) as,

$$I_{\text{dist}}(v_1, v_2) = d_{v_1 v_2}\,. \tag{6}$$

The usage of the above metric alongside with the k-split clustering algorithm produces sub-domains, where the distance of a node v that belongs at cluster $n \in \mathcal{N}$ from the other nodes within the same cluster n is always smaller than the distance of the same node v from any other node belonging to any of the rest of the clusters $\mathcal{N} \setminus \{n\}$.

Generally though, calculating similarity over a more detailed feature set captures characteristics more precisely and enables us to address the spatial variation of request rates $\boldsymbol{r}_v$. In this direction, we consider a hybrid similarity

metric based both on the topological distance as above, but also on the pairwise Euclidean distance of the content popularity at each node of the network, *i.e.,*

$$I_{\text{w.pop}}(v_1, v_2) = (1-w)\frac{d_{v_1 v_2}}{\max\limits_{v \in \mathcal{V}}(d_{v_1 v_2})} + w\frac{\left(\sqrt{\sum\limits_{m \in \mathcal{M}} \left(r_{v_1}^m - r_{v_2}^m\right)^2}\right)^{-1}}{\max\limits_{v \in \mathcal{V}}\left(\sqrt{\sum\limits_{m \in \mathcal{M}} \left(r_{v_1}^m - r_{v_2}^m\right)^2}\right)^{-1}}, \tag{7}$$

where $w \in [0, 1]$ is a weight for favouring either the relative topological distance between the nodes of the network or the relative Euclidean distance between the content popularity (request pattern) at those nodes. The usage of the Euclidean distance of content popularity transforms the Euclidean space of popularity into a metric space that can be combined with the topological distance of two nodes in the network.

Summarising, the k-split domain clustering algorithm enabling the above metric ($I_{\text{w.pop}}$) forms clusters where the nodes of each cluster are not only in close proximity to each other, but also groups users with similar request patterns as well. Below we present in pseudo-code the k-split clustering algorithm.

Algorithm 1 k-split Algorithm

Require: V: number of nodes
  k: number of clusters
  I: matrix with pairwise similarity metrics of nodes (v_1, v_2)
Ensure: The contents of each cluster K_i $\{i^{\text{th}}$ cluster$\}$
  v_j {node j}
  h_i {first node/head of cluster i}
  SET $h_1 = v_1$; $K_1 \leftarrow \{v_1, \ldots, v_V\}$
  for $l = 1$ to $k - 1$ **do**
    FIND $d \leftarrow max\{I(h_j, v_i) | v_i \in K_j$ and $1 \leq j \leq l\}$
    LET v_i the node that has the maximum distance d
    PLACE v_i to K_{l+1}
    SET $h_{l+1} = v_i$ {v_i is the first node/head of cluster $l + 1$}
    for $v_t \in (K_1 \cup \cdots \cup K_l)$ **do**
      LET j is the cluster that $v_t \in K_j$
      if $I(v_t, h_j) \geq I(v_t, v_i)$ **then**
        PLACE v_t from K_j to K_{l+1}
      end if
    end for
  end for
  RETURN $(K_1, \ldots, K_k)$

4.1.2. k-medoids domain clustering

We also make use of a second well-defined low-complexity algorithm for the partition of the domain. Particularly we use the k-medoids clustering algorithm [17], which attempts to minimise the distance between nodes belonging to the same cluster and

a node designated as the centre of that cluster. To run the k-medoids algorithm, we need to initially select the k nodes that will play the role of the centre of each cluster. Here we use the graph-based metric of betweenness centrality in order to find the k cluster centres. We remind that the betweenness centrality is an indicator of a node's centrality in a network. It is equal to the number of shortest paths from all vertices to all others that pass through that node. A node with high betweenness centrality has a large influence on the transfer of items through the network. After electing the k nodes with the largest betweenness centrality and assigning them as the centre of each cluster, we associate each one of the rest of the nodes $(V - k)$ to the closest medoid, based on their actual topological distance (*i.e.,* d_{uv}). If a node is equally close to more than one cluster centres, we assign it to the cluster with the smallest population of nodes.

Originally, the k-medoids algorithm (*a.k.a.* PAM algorithm) randomly select k out of the V nodes as the medoids and clusters the rest of the nodes according to the selected similarity metric. This procedure is followed iteratively for every combination of k out of V nodes and the configuration with the lowest cost is selected. Due to the exhaustive search approach though, the k-medoids algorithm can significantly affect the complexity of the network partitioning procedure and for that reason we used the betweenness centrality attribute of the nodes, for the election of the medoids as a way to further reduce its complexity. For comparison reasons we also present in the performance evaluation section another alternative for the election of the medoids. Particularly, we use the average distance of a node towards every other node in the network as the metric for the election of the k cluster centrers, but in general we left for future investigation the performance analysis of other variations of the used clustering algorithms, as well as other similarity metrics. Below we present in pseudo-code the k-medoids clustering algorithm.

Algorithm 2 k-medoids Algorithm

Require: V: number of nodes
  k: number of clusters
  I: matrix with pairwise similarity metrics of nodes (v_1, v_2)
  B: matrix with the betweenness centrality of nodes (or avr. distance)
Ensure: The contents of each cluster K_i $\{i^{\text{th}}$ cluster$\}$
  v_j {node j}
  m_i {first node/medoid of cluster i}
  for $l = 1$ to k **do**
    FIND $\underset{v_j}{\text{argmax}}(B)$ {find the nodes with the largest bet. centr.}
    SET $m_l = v_j$ and $B_j = 0$ {assign the medoids of the clusters}
    PLACE v_j to K_l
  end for
  for $j = 1$ to V **do**
    SET $D = [0, 0, \ldots, 0]$ {D has k entries}
    for $l = 1$ to k **do**
      SET $D(l) = I(m_l, v_j)$
    end for
    FIND $\underset{l}{\text{argmin}}(D)$ {the cluster that the node is closer to its medoid}
    PLACE v_j to K_l
  end for
  RETURN $(K_1, \ldots, K_k)$

Clustering of large data sets into subsets, where the members in a cluster are more

similar to each other than the members in different clusters, is a well investigated issue in the area of data mining. According to the method adopted to define clusters, the clustering algorithms can be broadly classified into four types (Partitional, Hierarchical, Density-based and Grid-based). For the purpose of this paper partitional clustering is the most suitable type, since the algorithms of this type attempt to determine an integer number of partitions that optimise a certain criterion function. Also, the two used members of this type (*i.e.,* k-means and k-medoids) are the most commonly used and they provide near optimal clustering, presenting at the same time low complexity of implementation. Of course, different clustering algorithms can be used for the partitioning of the domain, but due to space limitations this is left for future investigation. An overview of existing clustering schemes can be found in [50].

4.2. Content-to-cache mapping function

Once the sub-domains have been determined, an appropriate content-to-cache mapping function ρ should assign the content identifier of each item to a node of each sub-domain. As mentioned earlier, such function does not need to produce a cryptographic hash. In fact, it is desirable for its output to be produced with minimal processing. For example a modulo hashing of the content identifier is a suitable candidate and it is the basic hash function that we use in the performance evaluation section. Such a simplistic hash function allows the routers where the content requests are issued to forward them to the designated cache directly, without performing any lookup. In addition, this is performed without requiring any sort of inter-cache coordination since the hash function is computed in a distributed manner by the routers, thus achieving great scalability.

An ideal hash function should be capable of mapping content items to cache nodes so that the load of the caches is evenly spread. Also, an ideal hash function should incorporate the spatial characteristics of the content popularity at each region of the network and assign a content item as close as possible to the region of maximum interest. In that case a request for an item will not have to travel far within the network achieving smaller average retrieval latency. Also, by equally spreading the load across the network nodes, we can load-balance the network traffic and reduce significantly the stress of each link.

Unfortunately, a distributed hash function, like the modulo one, is not possible to achieve the above goals. To address these shortcomings we propose a new Bin packing-like content-to-cache assignment function whose objective is to evenly spread the load across the caches and at the same time assign content closer to the area of maximum interest. Of course, this is done at the expense of increased inter-cache coordination, since a coordination scheme should be adopted by the nodes of the network in order to compute the hash mapping matrix described previously in Section 3.

Bin packing content assignment function (Bin)

We remind that in a bin packing problem (*e.g.,* [51], [52]) items of different volumes (content popularity) should be assigned to a finite number of bins each of specific volume (equal load in our case). Particularly, we assume that a load of $\approx \frac{\sum_{v\in\mathcal{V}}\sum_{m\in\mathcal{M}} r_v^m}{V}$ requests should be assigned to each cache of a network of V nodes, assuming equal capacity among the nodes of the network. In order to enhance the efficiency of the proposed assignment function we allow each node v to accommodate items of total request capacity up to $R_v = (1+b)\frac{\sum_{v\in\mathcal{V}}\sum_{m\in\mathcal{M}} r_v^m}{V}$, where $b \geq 0$ is a trade-off between evenly spreading of the load and increased network performance (*i.e.,* smaller retrieval

latency or smaller link stress). So, the Bin packing assignment function assigns items to a cache of the network until a total of

$$R_v = \max\left\{ \max_{v\in\mathcal{V}, m\in\mathcal{M}} (\sum_{v\in\mathcal{V}} r_v^m), (1+b)\frac{\sum_{v\in\mathcal{V}}\sum_{m\in\mathcal{M}} r_v^m}{V} \right\} \tag{8}$$

requests have been assigned. Eq. (8) ensures that every item will be assigned to a cache even if the "even load distribution" requirement is violated. The first argument in the maximisation function of Eq.(8) ensures that in the case that an item is of very high popularity it can still be assigned to a node of the network.

The Bin packing assignment function is similar in rationale to the Greedy-global offline placement algorithm proposed in [53], which also uses workload information such as distance from the caches and request rates to assign content items to the caches. In particular, the newly proposed algorithm initially assumes that the hashing map $\boldsymbol{Y}(\rho)$ is empty. Then, at each iteration, the algorithm assigns each item $m \in \mathcal{M}$ to the node $u \in \mathcal{V}$ that yields the maximum average latency gain, starting from the most important one according to the item's relative popularity $g^m = \frac{\sum_{v\in\mathcal{V}} r_v^m}{\sum_{v\in\mathcal{V}}\sum_{i\in\mathcal{M}} r_v^i}$ (the item's total request rate over the total request rate of every content item in the network):

$$u = \arg\max \boldsymbol{G}^m \tag{9}$$

$$\text{where } \boldsymbol{G}^m = \{G_1^m, \ldots, G_V^m\}, \tag{9a}$$

$$\text{and } G_u^m = \sum_{v\in\mathcal{V}} r_v^m d_{vu}, \tag{9b}$$

$$\text{s.t. } \sum_{v\in\mathcal{V}}\sum_{m\in\mathcal{M}} r_v^m y_v^m \leq R_u. \tag{9c}$$

The algorithm continues for as long as not more that R_u requests (*i.e.,* Eq. (9c)) have been assigned to this node. The algorithm iterates until all the content identifiers have been assigned to exactly one cache. Note that, in contrast to a modulo-like assignment function, the Bin packing assignment does not necessarily assign the same number of items to each cache. Instead, items are assigned to each node, so that all caches are evenly loaded. The proposed algorithm could be either executed centrally by a centralized manager which collects all the necessary information from the nodes of the network, or it could be executed in a distributed manner where a set of cooperating managers are strategically placed in the network. This set of managers can be introduced in the network through a mechanism similar to the one presented in [54], but the actual realization of this approach is out of the scope of this paper.

With Bin packing assignment edge nodes are no longer able to resolve content-cache mappings by computing a hash function as in the case of modulo hashing. However, this can be still computed fast and with low memory requirements, for example using Bloom filters [55]. When the centralized manager (or the distributed managers) computes bin packing assignment offline, it also computes a set of Bloom filters, one per each caching node, indicating what content items each caching node is responsible for, and distribute them to edge nodes. An edge node can then identify the authoritative cache for a content by querying the Bloom filters for all the nodes in its cluster. Since each cluster only has a limited number of caching nodes, this operation is expected to be executed fast. In addition, it can also be easily accelerated by parallelization using SIMD processors, like GPUs. False positives can be addressed simply by sending an Interest to each caching node for which the lookup was successful.

The proposed Bin packing- assignment function is a centralized offline iterative algorithm which requires M (total number of items) iterations and has a computational complexity of $\mathcal{O}(VM)$ computations, where V is the total number of nodes in the examined domain. Note also, that the algorithm yields the optimum solution regarding the assignment of the identifiers to the nodes of the network since items are assigned only once at each domain (sub-domain in the clustering case) and no replicas of each item are allowed. It is obvious that the proposed Bin assignment function presents higher complexity compared to a modulo-like assignment function. In the evaluation section we examine both the modulo and the newly proposed assignment function and we provide a thorough comparison between them for every partitioning and routing scheme that we use (routing in a single domain or in the partitioned sub-domains) and for various network performance metrics; we examine how this additional complexity can further improve the performance of the hash routing framework.

5. Numerical Evaluation

5.1. Evaluation setup

In this Section, we evaluate through simulations the performance of hash routing in combination with the offline partitioning/clustering schemes presented in Section 4. We compare the performance of clustering techniques with the original hash-routing proposal [10] applied in the whole network/domain. Moreover, we examine two different content-to-cache assignment functions namely the modulo assignment function (*Mod*) and the newly proposed Bin packing assignment function (*Bin*). For comparison we also evaluate the performance of on-path content placement with opportunistic request to cache routing initially presented in [38]. According to this scheme, content items are cached as they travel through the network by every on-path cache and a request searches for a cached item on every cache along the resolution path. Table 1 depicts the whole spectrum of our methodology.

To carry out our performance evaluation we used *Icarus* [56], a discrete time event simulator for in-network caching and ICN environments. In the majority of the experiments presented here, we use a large network topology of $V = 110$ nodes (Interoute topology [57]) from the Internet Topology Zoo dataset [58], whereas in the last scenario we experiment with different real world topologies from the same dataset. The purpose of this experiment is to investigate whether the performance of the proposed hash routing and clustering schemes is affected by toplogical characteristics. We also assume that in each node of the network a total of $\lambda = 1$ request per second is generated. The requests at each node are generated according to a Poisson process and the corresponding λ parameter of the exponential distribution is equal to $\lambda = 1$. This means that at node v the request rate r_v^m for item m assuming that the Zipf exponent of the item's popularity is z and the item is ranked j-th out of the M information items within the Zipf distribution is given by,

$$r_v^m = \lambda \cdot \frac{1/j^z}{\sum_{i=1}^{M} 1/i^z}. \tag{10}$$

Thus, the request rate for each item at each node varies from 0-1 req/sec depending on item's popularity and ranking.

We consider a scenario where $M = 100K$ content items have to be assigned by the corresponding assignment functions. Recent measurement-based studies indicate that a small number of items often account for a large portion of traffic, especially

Table 1: Examined routing schemes.

Scheme	**HR**	**Assig. func.**	**Dom. Clust.**	**Metric**
Single domain HR	Yes	Bin pack	No	-
k-split dist. cl. HR	Yes	Bin pack	k-split	I_{dist}
k-split w. pop. cl. HR	Yes	Bin pack	k-split	$I_{\text{w.pop}}$
Bet. centr. cl. HR	Yes	Bin pack	k-medoids	I_{dist}
Avr. distance cl. HR	Yes	Bin pack	k-medoids	I_{dist}
On-path no-HR	No	-	No	-
Single domain HR mod.	Yes	Modulo	No	-
k-split dist. cl. HR mod.	Yes	Modulo	k-split	I_{dist}
k-split w. pop. cl. HR mod.	Yes	Modulo	k-split	$I_{\text{w.pop}}$
Bet. centr. cl. HR mod.	Yes	Modulo	k-medoids	I_{dist}
Avr. distance cl. HR mod.	Yes	Modulo	k-medoids	I_{dist}

for users located in certain areas (*e.g.,* a university campus [59]), or embedded in a social network [60]. This advocates that a small portion of the population of the items available in the network is actually requested. Additionally, the usage of Zipf distributions for the items' popularity implies that $100K$ items may account for the $\approx 95\%$ of the total demand considering an information space of approximately $500K$ items. On these grounds, our choice for the size of the items' population can be considered fair. This is further enhanced by recent results in [61], where authors analysed the traffic of a leading online video content provider in China (PPTV) over a two-week period with more than 196 million viewing instances involving more than 16 million users. Their findings show that, overall, these users watched around $500K$ unique videos and the distribution of user access to video content follows a Zipf distribution. The rest of the items not considered by the assignment function are assumed to be served directly from the origin server and are not considered in the following performance figures. We also assume that the background traffic generated in the network is negligible, and we left for future work the inclusion of this traffic in the process for the clustering of the network into sub-domains.

We also assume that the latency d_{ij} between two neighboring nodes $i, j \in \mathcal{V}$ and $(i, j) \in \mathcal{E}$ in the same administrative domain, typically ranges from a few up to 200 ms [62, 63], depending on the geographical coverage of the network.

For the evaluation of the proposed schemes we initially assume empty caches and for a period of $T = 6$hours (*warm up period*) we allow the system to populate the caches of the network. Afterwards, we monitor the performance of the network for a period of $T = 12$hours (*observation period*). Based on the request pattern observed during the warm up period we partition the domain offline (wherever the $I_{\text{w.pop}}$ metric is used Eq. 7) and assign the content identifiers to the caches (wherever the Bin assignment function is enabled). In the majority of the experiments we assume that the request pattern in the observation period is the same to the pattern in the warm up period, but we present in Section 5.5 a set of experiments where we have different patterns between the two periods in order to examine the resilience/robustness of the proposed schemes.

Our evaluation is based on the following metrics:

- The *Average Intra-domain Latency* (in *msec*) is the mean transfer time of a

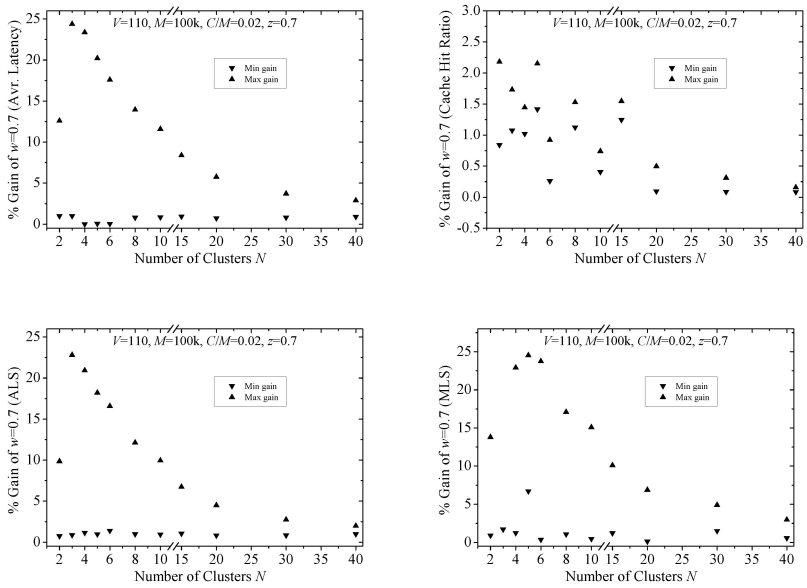


Figure 3: The performance gain (Min and Max) of weight value $w = 0.7$ in $I_{\text{w.pop}}$ over various values of w vs. the total number of clusters/sub-domains in the network.

content item from a network cache to the node where the request was generated.

- The *Cache Hit Ratio* is the ratio of the content requests that have found the requested item cached within the network, over the total number of requests issued during the observation period.
- The *Average Link Stress (ALS)* (in *items/sec*) is the mean number of delivered content items that travel through each link of the network.
- The *Maximum Link Stress (MLS)* (in *items/sec*) is the maximum number of delivered content items that travel through the most constrained/congested link of the network. This metric along with the Average Link Stress metric is indicative of the load balancing capabilities of each examined scheme.

In the schemes that use the the hybrid similarity metric $I_{\text{w.pop}}$, which incorporates both the topological distance and the request pattern of the nodes of the network we assume that the weight w in Eq. (7) is equal to $w = 0.7$. Due to space limitations we cannot depict the performance of this metric for various values of w, but based on the given setup we found that this value ($w = 0.7$) performs best among all considered performance metrics. In particular, we depict in Figure 3 the performance gain (Min and Max) using parameter value $w = 0.7$ over twenty different values for w and for various values regarding the total number of clusters/sub-domains in the network. Also, we depict in the following figures only the greyed schemes from Table 1. The greyed schemes that enable hash routing (HR) use the newly proposed Bin assignment function, which generally behaves better than the modulo assignment function. To highlight the extra merits of the Bin assignment function, we also present an one-to-one comparison of every examined scheme that uses the Bin function against the counterpart scheme that uses the modulo function. Finally, for the schemes that enable the Bin assignment function we assume that the parameter b in Eq.(8) is equal to $b = 0.05$, just to ensure that all items have been assigned in a cache of the network.

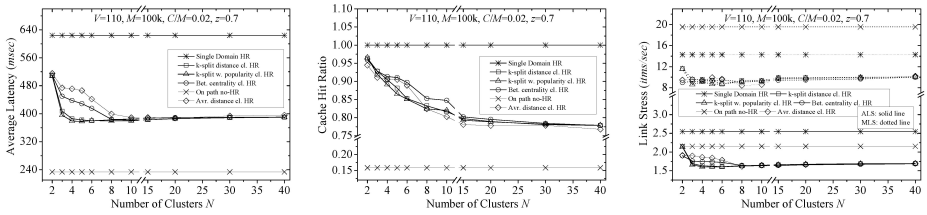


Figure 4: The performance of the examined routing schemes vs. the total number of clusters/sub-domains in the network.

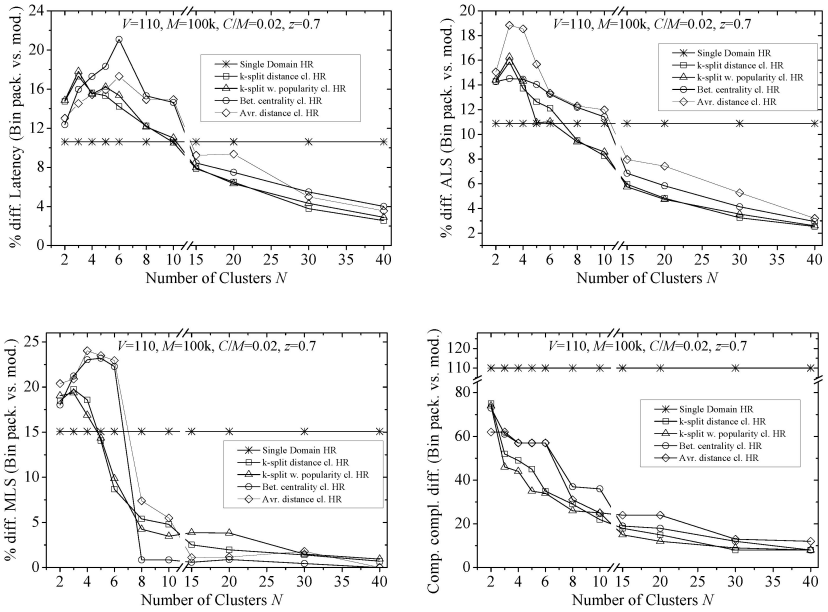


Figure 5: The performance difference of the Bin assignment function over the modulo assignment function vs. the total number of clusters/sub-domains in the network.

We leave for future investigation the examination of the impact of parameter b in the performance of the network (improved retrieval latency over uneven load between the caches).

5.2. Impact of the number of clusters

Figure 4 depicts the impact of the number of clusters N, that the domain is partitioned in, on the performance of the network. Note that when a request within a cluster does not find a cached item at the node which is responsible for the corresponding part of the hash space it always heads towards the origin node. Another alternative is to assume that a request is sent to other clusters of the same domain in order to increase the cache hit ratio of the system. Such an approach though, raises scalability issues since a node should keep track of the cache assignment in every other sub-domain. Also, due to the probabilistic nature of caching in ICN this might explode the transfer time of an item, since there is no guarantee that an item will be found in any other cluster.

Obviously, the schemes that are applied in a clustered domain are the only ones affected by the number of clusters. The other two depicted schemes (*i.e.,* Single Domain HR and On-path no-HR) are presented for comparison reasons. From Figure 4 we observe that a small number of clusters (3-5) can achieve a reduction in the average retrieval latency up to 60% compared to the single domain scenario at the cost of a less than 5% cache misses. On the other hand, every scheme that assumes hash routing and off path caching performs up to eight times better than the simplistic on-path content placement with opportunistic request to cache routing scheme. Of course, the on path scheme retrieves the items cached in the domain faster than off-path caching schemes; this, however, is the case only for 16% of the issued requests, whereas the remaining 84% is fetched from the origin node. Furthermore, the schemes that involve clustering manage to better spread/balance the load of the network throughout its links, performing up to 60% better from the single domain cases in terms of both the average and the maximum link stress of the network. By partitioning the domain into clusters we manage to restrict the requests from travelling far in the network and despite the fact that each node is responsible for more items than in the single domain hash-routing case, the usage of the proposed clustering schemes allows the network to utilise better its resources with only a small penalty in terms of cache hit ratio.

From the point of view of the different clustering algorithms and the corresponding clustering metrics (again in Fig. 4), we observe that the k-medoids algorithm (*i.e.,* Bet. centr. cl. HR and Avr. distance cl. HR schemes) behave slightly worse both in terms of retrieval latency and in terms of the cache hit ratio for small number of clusters than the k-split counterpart. However, its performance is superior when it comes to the load balancing metrics for more clusters (similar regarding the delay and the cache hit ratio). Note that since the Bet. centr. cl. HR scheme performs better than the Avr. distance cl. HR scheme for every conducted experiment we only depict the former one in the rest of evaluated scenarios.

In Figure 5 we depict the percentage difference of the newly proposed Bin packing assignment function against the modulo assignment function for every scheme that incorporates the hash routing technique. Note that both functions perform exactly the same in terms of the cache hit ratio metric and for that reason we depict only their one-to-one comparison for the rest of the metrics. From Fig. 5, we observe that the Bin assignment function behaves up to 25% better than the modulo at the cost of increased complexity and coordination between the nodes. As the number of clusters increases the number of nodes within each cluster decreases minimising in that case the extra merits of the Bin function. For instance, when $N = 30$, the largest cluster has only 6 nodes which implies that the assignment function has limited impact on the performance of such small sub-domains.

In Figure 5 we also depict the computational complexity gap of the Bin packing assignment function over the modulo one. In more details, we observe that for small number of clusters (*e.g.,* 5 sub-domains) the complexity of the Bin function is up to fifty times larger that the complexity of the modulo function. This complexity gap decreases as we increase the number of clusters, but as mentioned above, so do the merits of Bin function. This figure may also serve as a benchmark for a network manager to decide which assignment function to enable, since the off-line Bin function can significantly improve the performance of the network at a significant higher computational cost.

Takeaway point(s): *A small number of clusters can reduce the average retrieval latency up to* 60% *with only a small penalty of less than* 5% *in the cache hit ratio.*

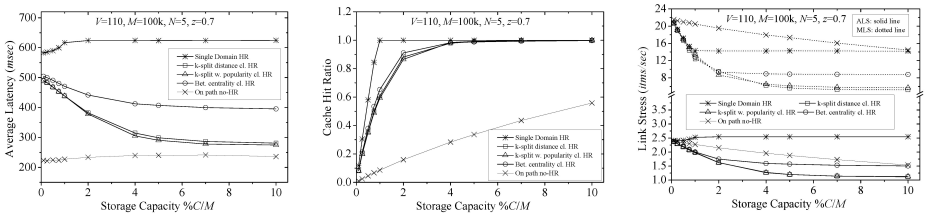


Figure 6: The performance of the examined routing schemes vs. the storage capacity of each ICN router.

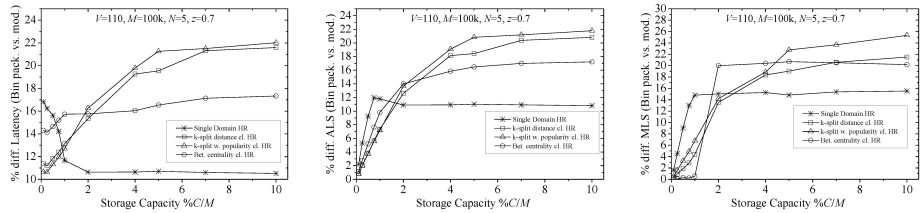


Figure 7: The performance difference of the Bin assignment function over the modulo assignment function vs. the storage capacity of each ICN router.

5.3. Impact of the cache size

In Figure 6, we depict the impact of the cache capacity on the performance of the routing schemes, expressed as the fraction of the content population that can be cached at each node. We also assume that the domain is split into five clusters for the schemes where partitioning of the domain is applied. Generally, we observe the same findings to the previous experiment where the clustered domain schemes perform up to 50% better in terms of the mean retrieval delay of an item. Also, we observe an exponential increase of the cache hit ratio of those schemes and for small cache capacities they perform less than 5% worse than the Single domain hash routing scheme. The on-path routing scheme follows a linear increase in the cache hit ratio with regards to the increase in the cache capacity of each node, but it always performs significantly worse compared to the hash routing scheme. In particular, for small values of the cache capacity ($\leq$ 2% of the content population) the corresponding hash routing schemes perform up to 10 times better than on-path caching.

The three clustering schemes perform up to 50% better than the single domain scheme in terms of the average link stress of the network, and up to 60% better in terms of the maximum link stress. This performance gap combined with the high cache hit ratio of the domain-clustering schemes (for the cases where the cache capacity of a node is $\geq$ 2% of the total item population) shows that partitioning of the domain into smaller domains increases the Quality of Experience of users (*i.e.,* results in low retrieval latency). At the same time, the inter-domain traffic is minimsed, an issue which is of primary importance to the ISP, in order to reduce its transit costs.

Figure 7, similarly to Figure 5, shows that the Bin function performs up to 25% better than the modulo assignment function. Even for very small cache capacities, the new assignment function that incorporates the spatial characteristics of the request pattern achieves a significant decrease in the mean retrieval latency (more than 10%). As discussed before, this performance improvement comes at the cost of extra inter-cache coordination. Of course in the cases where this extra cost is not tolerable the simplistic and distributed modulo assignment function is still a reliable solution since it constantly performs well with respect to every examined performance metric.

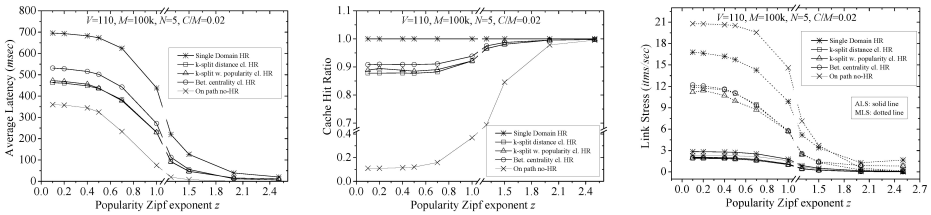


Figure 8: The performance of the examined routing schemes vs. the exponent of the content popularity distribution.

Takeaway point(s): *For small cache capacities, typical in real world scenarios, the proposed hash routing scheme performs up to ten times better than conventional on path caching. Furthermore, the partitioning of the domain further improves the users' perceived QoE and the load balancing capabilities of the network.*

5.4. Impact of the popularity distribution

In the above scenarios we assumed a specific value for the Zipf exponent of the items' popularity. Measurement-based studies, such as [40], suggest that the Zipf exponent z for web traffic lies in the range of $0.64 - 0.84$, while other types of traffic (*e.g.,* P2P or video) may follow different popularity patterns [41]. In particular, in [61] authors found that the distribution of the user access to video content is a Zipf-like with exponent parameter $z = 1.174$.

In this scenario, we examine a wider range of values for the Zipf distribution. The results are presented in Fig. 8. Particularly, for small values of z we observe a behaviour similar to the previous two experiments. For $z \geq 1.3$ the set of the items that account for the majority of the requests is very small and those items tend to remain cached for the whole duration of the observation period. More in particular, when $z = 1.5$ there is a small number of items ($\approx 0.005M$) that account for the $\approx 97\%$ of the total requests. Those items can almost always be found within the domain caches. Furthermore, due to the spatial characteristics of the request pattern, the usage of our sophisticated Bin packing assignment function allows a request to find them closer to the requesting node, as shown in Figure 9. Particularly, from Figure 9 we observe that when $z \geq 1.3$ the Bin function performs up to 97% better than the modulo function. Generally, the modulo function is not affected by the Zipf distribution of the request pattern but at the same time it can not exploit its unique spatial characteristics. Of course, as explained previously, there is a trade-off between the actual performance of the network and the scalability of the assignment function in use. This means that a network manager might prefer a centralised offline complex assignment function for specific applications (like video distribution), as compared to a simple modulo assignment function, since this will improve performance at the cost of extra co-ordination.

Takeaway point: *The newly proposed Bin packing assignment function can dramatically improve the performance of the content retrieval process, when applied in systems where the popularity distribution of the content items is less uniform (e.g., online video content providers).*

5.5. Impact of content popularity dynamics

So far, we have assumed that the content popularity between the warm up and the observation period is stable and based on the request pattern observed during

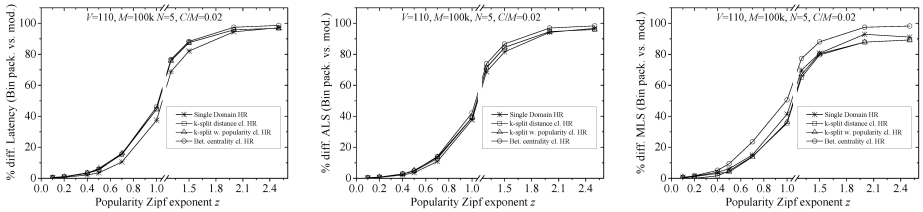


Figure 9: The performance difference of the Bin assignment function over the modulo assignment function vs. the exponent of the content popularity distribution.

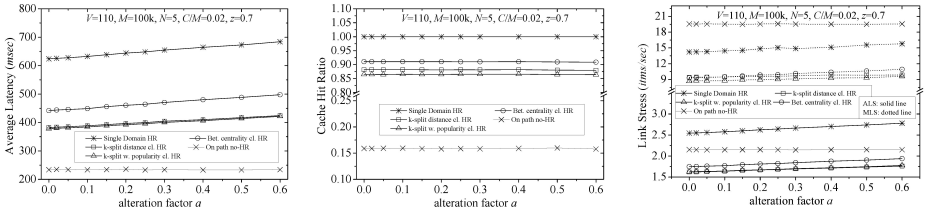


Figure 10: The performance of the examined routing schemes vs. the content popularity alteration factor.

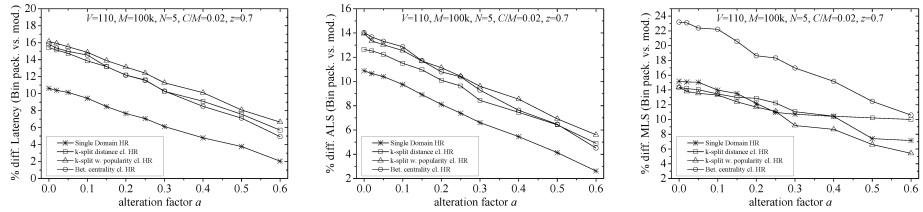


Figure 11: The performance difference of the Bin assignment function over the modulo assignment function vs. the content popularity alteration factor.

the warm up period we partitioned the domain (wherever the $I_{\text{w.pop}}$ metric is used) and assigned the content identifiers to the caches (wherever the Bin function is used). Generally, content popularity tends to change over time. In this section we assume that the request pattern observed in the warm up period (and used for the partition and the content assignment) is different to the pattern observed during the observation period. We model this modification of request vectors through a popularity alteration factor a. We assume that the ranking of the items within the Zipf popularity distribution at each node is altered by a factor a; $a \cdot M$ items have a different ranking at the beginning of the observation period than the one at the end of the warm up period.

Figure 10 depicts the impact of this factor on the performance of the examined schemes. From Figure 10 we observe that this alteration factor has limited impact on the performance of the examined schemes. Although this was an expected result for the cases of modulo assignment and the clustering schemes that use the I_{dist} metric, we expected some differences in the case of the scheme that incorporates the request pattern for the partitioning of the domain (*i.e.,* k-split w. pop. cl. HR). Nonetheless, even in case of "k-split w. pop. cl. HR", we only observe a very small impact in the network performance even if the ranking of the items has changed up to 60%. This performance degradation is less than 8% even for the most vulnerable scheme (*i.e.,* k-split w. pop. cl. HR).

Of course, as depicted in Figure 11, outdated knowledge of the request pattern tends to minimise the extra merits of the Bin assignment function, but in all cases,

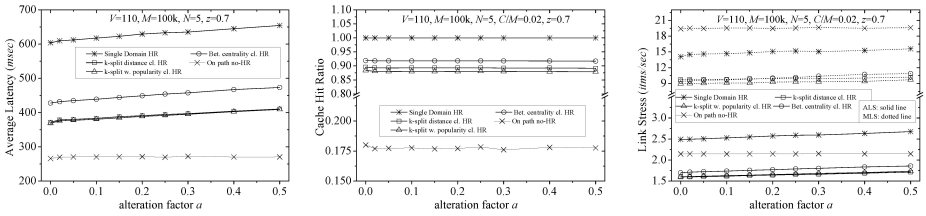


Figure 12: The performance of the examined routing schemes vs. the content popularity alteration factor for non uniform storage capacities of the caches and non uniform alteration factor at each node.

the schemes that make use of the Bin packing assignment function still perform better than those schemes that use modulo assignment. Definitely though, an enhanced performance of $\approx 2\%$ in a volatile environment might not be enough to justify a centralised and rather complex function as well as a more sophisticated clustering scheme instead of a scalable assignment function and a rather simplistic domain partitioning scheme. From both figures we deduce that the proposed hash routing technique (applied in the whole domain or in clustered partitions) is robust against inaccurate estimation of content popularity and can still operate efficiently even with less accurate estimates of content popularity, which is a major advantage, given the difficulty of such estimations.

In order to examine the performance of hash routing and the corresponding assignment functions to a more realistic environment, we repeated the above experiment in a network where the nodes have non equal storage capacity and the alteration factor is different at each node. Particularly, we assumed that the capacity of each node in analogous to its degree (number of edges). Assuming that g_v is the degree of node v and $\mathcal{C}$ is the total capacity of the network, then the storage capacity of node v is $C_v = \mathcal{C}\left\lceil \frac{g_v}{\sum_{v\in\mathcal{V}} g_v} \right\rceil$. Also, we assumed that the ranking of the items within the Zipf popularity distribution at each node is altered by a random factor a'; $a' \cdot M$ items have a different ranking at the beginning of the observation period than the one at the end of the warm up period, where $a' \in [-a, a]$. The results are depicted in Figure 12 and are in perfect alignment with the results of Figure 10, which implies that hash routing and the proposed assignment functions can operate under less accurate estimates of content popularity in every environment (*i.e.,* non uniform storage capacity and non equal changes in popularity).

Takeaway point(s): *The proposed assignment functions and the proposed domain partitioning techniques are robust against inaccurate estimations of the content popularity and can be executed in larger time intervals without significant loss in their inherent merits.*

5.6. Impact of the topology size

We have so far experimented using the same real world network topology and examined the performance of the proposed routing mechanisms for various system parameters. In this section, we examine the performance of one of the proposed routing schemes (k-split dist. clustering hash routing algorithm) against the Single domain HR algorithm for various network topologies from the Internet Zoo dataset. Figure 13 depicts the performance of the examined routing scheme over the single domain HR scheme for ten different topologies and for various clustering approaches (*i.e.,* number of clusters). From Figure 13 we observe that the examined clustering

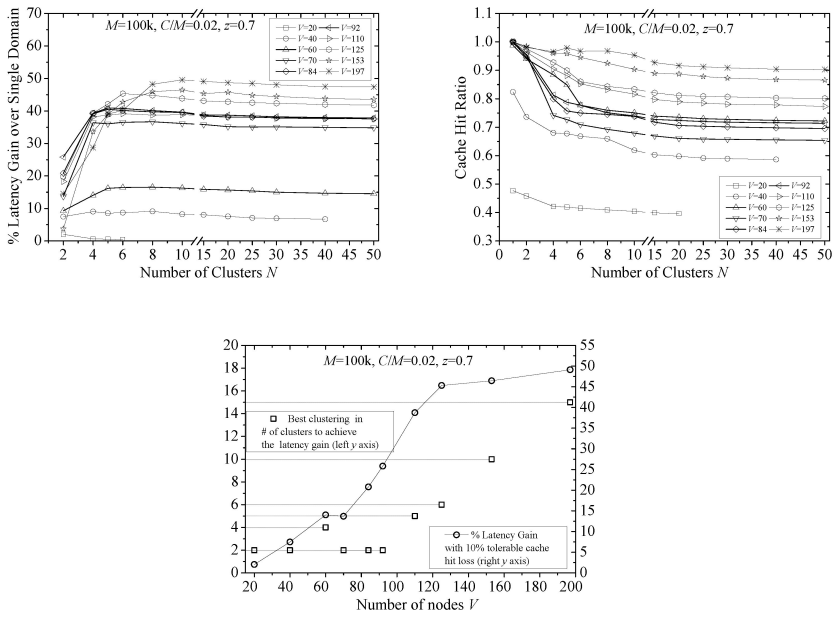


Figure 13: The performance difference (gain) of the k-split dist. cl. HR algorithm over the Single Domain HR algorithm for various network topologies from the Zoo dataset and for various number of clusters/sub-domains in the network. The last plot shows the maximum average retrieval latency gain that it can be achieved for each topology and the corresponding clustering scheme (sub-domains) assuming a tolerable 10% loss in cache hit compared to the single domain case.

scheme significantly improves the average retrieval latency in every examined topology with a small penalty in the cache hit ratio. This improvement is subject to the actual characteristics of the underlying topology and can vary from 2% up to 50% in terms of decrease in retrieval latency.

The last plot of Figure 13 depicts the maximum performance gain of using the proposed clustering and hash routing scheme over the Single domain HR when the network manager can tolerate up to 10% loss in the cache hit ratio. In the same plot we also depict the clustering degree (*i.e.,* number of clusters) under which this performance gain can be achieved. From this plot it is clear that the network topology characteristics severely influence the performance of the used routing scheme. At the same time, however, it is also clear that it is difficult to find a closed form formula for the clustering approach that best fits a given network topology, as this depends on the specific characteristics of each topology. For example the best performance gain for the topology with the 60 nodes is achieved when it is clustered in four partitions, whereas the topology with the 92 nodes performs best when partitioned in two sub-domains.

By and large, for topologies of 100 nodes or more, there is a clear increasing trend of the optimal number of clusters needed to achieve the best possible performance. In contrast, for domains of less than 100 nodes the situation is not clear. For example, for a group of topologies comprising of 20, 40, 70 and 84 nodes, two clusters is the optimal setting, while for the topology of 60 nodes, four clusters is the optimal and for a topology of 92 nodes, two clusters are again the optimal clustering scheme.

Takeaway point(s): *Hash routing combined with domain partitioning can signifi-*

cantly improve the performance of the network but the used clustering scheme should be carefully coupled with the unique characteristics of each network topology.

6. Conclusions

The process of resolving requests to in-network caches has concerned the ICN community so far and has resulted in several proposals to deal with this issue. However, the tradeoff between performance (in terms of cache hits) and co-ordination overhead (which raises scalability concerns) is not easy to balance. We believe that hash-routing techniques offer a very easy to implement, efficient and scalable way of assigning content items to network caches and redirecting content requests to the corresponding cache. Our initial results in [10] revealed significant increase in performance, but did not take into account the case of very large networks, where the stretch of the detour path can become very big.

In this paper we have extended our previous study to deal with this issue of extensive detour trips indicated by the hash-routing function. Our proposal builds on nodal clustering techniques that limit the number of nodes that a request will have to travel in order to reach the corresponding cache in case of very large networks. We report significant reduction in delivery delay with a slight decrease in cache hit rate. In general hash routing combined with domain partitioning can reduce the average item delivery latency up to 50% depending on the various system parameters (*i.e.,* popularity pattern, cache size, content popularity fluctuations), as well as on the actual topological characteristics of the network itself.

The proposed framework requires that ICN architectures adopt hash-routing as their routing mechanism. On the other hand, and in order to maintain the inherent characteristics of the on path content placement with opportunistic request-to-cache routing, the approach presented could be also applied to a partially coordinated caching scheme similar to the one in [63]. In such a scheme, a fraction of each router's cache could be used to support the proposed hash-routing framework and the rest of the cache capacity could be used opportunistically to cache passing-by items. In this case, an interest packet that heads either towards the router indicated by the hash assignment function or towards the content server, searches at each intermediate router for a matching cached information item. We intend to explore such a hybrid scheme in our future work in this area.

References

[1] Cisco visual networking index: Forecast and methodology, `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html`.

[2] D. Kutscher, S. Eum, K. Pentikousis, I. Psaras, D. Corujo, D. Saucez, T. Schmidt, M. Waehlisch, ICN research challenges, IRTF, 2014.

[3] I. Psaras, W. K. Chai, G. Pavlou, In-network cache management and resource allocation for information-centric networks, IEEE Transactions on Parallel and Distributed Systems pp. 2920–2931, 2014.

[4] K. Cho, M. Lee, K. Park, T. Kwon, Y. Choi, S. Pack, Wave: Popularity-based and collaborative in-network caching for content-oriented networks, IEEE INFOCOM WKSHPS, pp. 316–321, 2012.

[5] K. W. Ross, Hash-routing for collections of shared web caches, IEEE Network, pp. 37–44, 1997.
[6] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, D. Lewin, Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web, ACN STOC, pp. 654–663, 1997.
[7] Y. Wang, K. Lee, B. Venkataraman, R. Shamanna, I. Rhee, S. Yang, Advertising cached contents in the control plane: Necessity and feasibility, IEEE INFOCOM WKSHPS, pp. 286–291, 2012.
[8] S. Eum, K. Nakauchi, M. Murata, Y. Shoji, N. Nishinaga, Catt: Potential based routing with content caching for icn, ACM ICN Workshop, pp. 49–54, 2012.
[9] H.-g. Choi, J. Yoo, T. Chung, N. Choi, T. Kwon, Y. Choi, Corc: Coordinated routing and caching for named data networking, ACM/IEEE ANCS, pp. 161–172, 2014.
[10] L. Saino, I. Psaras, G. Pavlou, Hash-routing schemes for information centric networking, ACM ICN Workshop, pp. 27–32, 2013.
[11] T. Berners-Lee, R. Fielding, L. Masinter, Uniform Resource Identifier (URI) Generic syntax, RFC 3986, 2005.
[12] S. Farrell, D. Kutscher, C. Dannewitz, B. Ohlman, A. Keranen, P. Hallam-Baker, Naming things with hashes, RFC 6920, 2013.
[13] W. K. Chai, D. He, I. Psaras, G. Pavlou, Cache "less for more" in information-centric networks (extended version), Computer Communications, pp. 758 –770, 2013.
[14] V. Sourlas, L. Gkatzikis, P. Flegkas, L. Tassiulas, Distributed cache management in information-centric networks, IEEE Transactions on Network and Service Management, pp. 286–299, 2013.
[15] Y. Chen, L. Qiu, W. Chen, L. Nguyen, R. Katz, Efficient and adaptive web replication using content clustering, IEEE JSAC pp. 979–994, 2003.
[16] T. F. Gonzalez, Clustering to minimize the maximum intercluster distance, Theoretical Computer Science, pp. 293–306, 1985.
[17] L. Kaufman, P. Rousseeuw, Clustering by Means of Medoids, Reports of the Faculty of Mathematics and Informatics, Faculty of Mathematics and Informatics, 1987.
[18] P. K. Agyapong, M. Sirbu, Economic incentives in information-centric networking: implications for protocol design and public policy, IEEE Communications Magazine, pp. 18–26, 2012.
[19] G. Dán, Cache-to-cache: Could ISPs cooperate to decrease peer-to-peer content distribution costs?, IEEE Transactions on Parallel and Distributed Systems, pp. 1469–1482, 2011.
[20] V. Pacifici, G. Dán, Content-peering dynamics of autonomous caches in a content-centric network, IEEE INFOCOM, pp.1079–1087, 2013.
[21] F. Kocak, G. Kesidis, T.-M. Pham, S. Fdida, The effect of caching on a model of content and access provider revenues in information-centric networks, IEEE SocialCom, pp. 45–50, 2013.
[22] T.-M. Pham, S. Fdida, P. Antoniadis, Pricing in information-centric network interconnection, IFIP Networking, pp. 1–9, 2013.
[23] A. Araldo, D. Rossi, F. Martignon, Cost-aware caching: Caching more (costly items) for less (ISPs operational expenditures), IEEE Transactions on Parallel and Distributed Systems, 2015.
[24] S. Borst, V. Gupta, A. Walid, Distributed caching algorithms for content distribution networks, IEEE INFOCOM, pp. 1–9, 2010.

[25] I. D. Baev, R. Rajaraman, Approximation algorithms for data placement in arbitrary networks, ACM-SIAM Symposium on Discrete Algorithms, pp. 661–670, 2001.

[26] V. Sourlas, P. Flegkas, G. S. Paschos, D. Katsaros, L. Tassiulas, Storage planning and replica assignment in content-centric publish/subscribe networks, Computer Networks, pp. 4021 – 4032, 2011.

[27] J. Li, H. Wu, B. Liu, J. Lu, Y. Wang, X. Wang, Y. Zhang, L. Dong, Popularity-driven coordinated caching in named data networking, ACM/IEEE ANCS, pp. 15–26, 2012.

[28] G. Tyson, S. Kaune, S. Miles, Y. El-khatib, A. Mauthe, A. Taweel, A trace-driven analysis of caching in content-centric networks, ICCCN, pp. 1–7, 2012.

[29] S. Guo, H. Xie, G. Shi, Collaborative forwarding and caching in content centric networks, IFIP Networking, pp. 41–55, 2012.

[30] M. Lee, K. Cho, K. Park, T. Kwon, Y. Choi, Scan: Scalable content routing for content-aware networking, IEEE ICC, pp. 1–5, 2011.

[31] I. Psaras, K. Katsaros, L. Saino, G. Pavlou, LIRA: A Location-Independent Routing Layer Based on Source-Provided Ephemeral Names, UCL Tech. Rep. 2015.

[32] V. Sourlas, P. Flegkas, L. Tassiulas, A novel cache aware routing scheme for information-centric networks, Computer Networks, pp. 44 – 61, 2014.

[33] G. Rossini, D. Rossi, Coupling caching and forwarding: Benefits, analysis, and implementation, ACM ICN, pp. 127–136, 2014.

[34] L. Wang, S. Bayhan, J. Ott, J. Kangasharju, A. Sathiaseelan, J. Crowcroft, Prodiluvian: Understanding scoped-flooding for content discovery in information-centric networking, ACM ICN, pp. 9–18, 2015.

[35] R. Tewari, M. Dahlin, H. Vin, J. Kay, Design considerations for distributed caching on the internet, IEEE Distributed Computing Systems, pp. 273–284, 1999.

[36] D. Wessels, K. Claffy, ICP: Internet cache protocol, RFC 2186, 1997.

[37] S. Bhattacharjee, K. Calvert, E. Zegura, Self-organizing wide-area network caches, IEEE INFOCOM, pp. 600–608, 1998.

[38] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, R. L. Braynard, Networking named content, ACM CoNEXT, pp. 1–12, 2009.

[39] T. Janaszka, D. Bursztynowski, M. Dzida, On popularity-based load balancing in content networks, ITC, pp. 1–8, 2012.

[40] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker, Web caching and Zipf-like distributions: evidence and implications, IEEE INFOCOM, pp. 126–134, 1999.

[41] G. Dán, N. Carlsson, Power-law revisited: Large scale measurement study of P2P content popularity, USENIX IPTPS, 2010.

[42] S. K. Fayazbakhsh, Y. Lin, A. Tootoonchian, A. Ghodsi, T. Koponen, B. Maggs, K. Ng, V. Sekar, S. Shenker, Less pain, most of the gain: Incrementally deployable ICN, ACM SIGCOMM, pp. 147–158, 2013.

[43] V. Sourlas, G. S. Paschos, P. Mannersalo, P. Flegkas, L. Tassiulas, Modeling the dynamics of caching in content-based publish/subscribe systems, ACM SAC, pp. 478–485, 2011.

[44] I. Psaras, R. G. Clegg, R. Landa, W. K. Chai, G. Pavlou, Modelling and Evaluation of CCN-Caching Trees, IFIP Networking, pp. 78–91, 2011.

[45] H. Che, Y. Tung, Z. Wang, Hierarchical web caching systems: modeling, design and experimental results, IEEE JSAC, pp. 1305–1314, 2002.

[46] C. Fricker, P. Robert, J. Roberts, A versatile and accurate approximation for LRU cache performance, ITC, pp. 1–8, 2012.

[47] V. Martina, M. Garetto, E. Leonardi, A unified approach to the performance analysis of caching systems, IEEE INFOCOM, pp. 2040–2048, 2014.

[48] N. Laoutaris, A closed-form method for LRU replacement under generalised power-law demand.
URL `http://arxiv.org/abs/0705.1970`

[49] D. Aloise, A. Deshpande, P. Hansen, P. Popat, NP-hardness of Euclidean sum-of-squares clustering, Machine Learning, pp. 245–248, 2009.

[50] M. Halkidi, Y. Batistakis, M. Vazirgiannis, On clustering validation techniques, Intelligent Information Systems, pp. 107–145, 2001.

[51] M. R. Garey, R. L. Graham, J. D. Ullman, An analysis of some packing algorithms, Combinatorial Algorithms pp. 39–47, 1973.

[52] E. G. Coffman, Jr., M. R. Garey, D. S. Johnson, Approximation algorithms for np-hard problems, PWS Publishing, pp. 46–93, 1997.

[53] J. Kangasharju, J. Roberts, K. W. Ross, Object replication strategies in content distribution networks, Computer Communications, pp. 376–383, 2002.

[54] R. Clegg, S. Clayman, G. Pavlou, L. Mamatas, A. Galis, On the selection of management/monitoring nodes in highly dynamic networks, IEEE Transactions on Computers, pp. 1207–1220, 2013.

[55] H. Lim, K. Lim, N. Lee, K.-H. Park, On adding bloom filters to longest prefix matching algorithms, IEEE Transactions on Computers, pp. 411–423, 2014.

[56] L. Saino, I. Psaras, G. Pavlou, Icarus: a caching simulator for Information Centric Networking, ICST, pp. 66–75, 2014.

[57] The Interoute topology, `http://www.topology-zoo.org/maps/Interoute.jpg`. [Online; accessed 02-June-2015] (2010).

[58] S. Knight, H. Nguyen, N. Falkner, R. Bowden, M. Roughan, The internet topology Zoo, IEEE JSAC, pp. 1765–1775, 2011.

[59] M. Zink, K. Suh, Y. Gu, J. Kurose, Characteristics of YouTube network traffic at a campus network: Measurements, models, and implications, Computer Networks, pp. 501–514, 2009.

[60] X. Bao, Y. Lin, U. Lee, I. Rimac, R. Choudhury, Dataspotting: Exploiting naturally clustered mobile devices to offload cellular traffic, IEEE INFOCOM, pp. 420–424, 2013.

[61] Y. Sun, S. K. Fayaz, Y. Guo, V. Sekar, Y. Jin, M. A. Kaafar, S. Uhlig, Trace-driven analysis of ICN caching algorithms on video-on-demand workloads, ACM CoNEXT, pp. 363–376, 2014.

[62] K. C. Kang, K.-D. Nam, D. J. Jeon, S. Y. Kim, Delay characteristics of high-speed internet access network, APNOMS, 2003.

[63] Y. Li, H. Xie, Y. Wen, Z.-L. Zhang, Coordinating in-network caching in content-centric networks: Model and analysis, IEEE ICDCS, pp. 62–72, 2013.