Local cooperative caching policies in multi-hop D2D networks

Local cooperative caching policies in multi-hop D2D networks

Javed Iqbal, Paolo Giaccone, Claudio Rossi
Department of Electronics and Telecommunications
Politecnico di Torino, Italy
Email: {javed.iqbal, paolo.giaccone,claudio.rossi}@polito.it

***Abstract*—Cooperative caching schemes allow to improve the performance of multi-hop networks based on device-to-device (D2D) communications. Indeed, each node does not only share its transmission capabilities to physically extend the network, but it also shares its storage to cache copies of contents for the sake of other nodes. It results in an increased network performance for users, since caching decreases both network load and latency to reach a content.**

The design of effective caching policies in a network of caches is very challenging and all the known solutions must be adapted both to the topology and to the request traffic pattern. In this paper, we consider a linear topology, representing a sequence of adjacent nodes, investigating the performances of both local and distributed cooperative caching policies. We specifically investigate *where* to apply the caching policy. Interestingly, we show that a simple local caching policy, that caches only the contents requested by the node itself, is not worse (or even better) than distributed policies, in which the content is eventually cached across the path from the requester node to the closest copy of the content. In some sense, we show that simplicity pays off.

I. Introduction

Device-to-device (D2D) communications are enabled by recent layer-2 technologies, e.g., WiFi Direct [1], and they are becoming more and more available in modern mobile devices like smartphones and tablets. Besides the simplicity to support direct communications between nodes, these technologies are evolving to support also multi-hop communications, which are obtained by a sequence of D2D hops, requiring the cooperation of the intermediate nodes which relay the traffic for end nodes. Moreover, new specific middlewares (like AllJoyn [2]) ease the development of applications tailored for such scenario, enabling peer-to-peer capabilities among terminal nodes, without the need of a centralized access point (as in legacy 802.11). Furthermore, the D2D paradigm is also supported in new LTE Direct standard, specified in 3GPP (Release 12) [3].

Furthermore, modern mobile devices are equipped with large storage capacity that could be in the order of several gigabytes. A considerable amount of such storage can be considered as a free resource nowadays, and users may prefer to share it more than other constrained and expensive resources, like CPU or memory, since storage availability does not directly affects Quality of Experience of running applications. This fact enables the adoption of cooperative caching schemes, in which each mobile device acts as a caching node. Each node may store locally all the received contents in its cache, including the ones to be forwarded in a multi-hop fashion. Content distribution across nodes eventually allows a user to retrieve desired contents in its close proximity. Thus, cooperative caching can reduce the delay to access contents, bringing a benefit to users, as well as the network load at the infrastructure, bringing a benefit to wireless network operators. Note that also the new network paradigm, denoted as Information Centric Networking [4], is typically based on cooperative caching occurring at each node and is one of the motivating scenarios for our work.

A cooperative caching strategy is usually defined in terms of the caching eviction policy, i.e. when the cache becomes full, the node must choose some content to remove from the cache to admit the new content. We instead focus our investigation on *where* to cache (i.e. the insertion policy), considering as candidates for caching all the nodes along the path from the requester to the closest content provider. As extreme cases, caching could occur in all the nodes along the path, or just in the requester node. The former case refers to a *distributed cooperative caching* scheme, in which all the nodes are storing contents requested by other nodes. The latter case refers instead to a *local cooperative caching* scheme in which a content is stored only at the local cache where it has been requested. Note that this policy can still be defined as cooperative, since each node is available to provide a copy of its cached contents to other nodes, but it stores only contents requested by itself.

In our work we will compare local and distributed cooperative caching schemes, under different synthetic content request models as well as with real content request traces obtained by an Italian ISP.

The rest of the paper is organized as follows. In Sec. II, we describe the system model considered in our investigation. In Sec. III we discuss the related works. In Sec. IV we explain the specific caching policies adopted for the comparison. Sec. V explains the methodology adopted to get the results shown in Sec. VI. Finally, in Sec. VII we draw our conclusions.

II. Cooperative caching

We consider a wireless network consisting of a *server* and N nodes interconnected through a multi-hop linear topology, as shown in Fig. 1. The choice of this topology is somehow arbitrary, and besides its simplicity it can be considered as one important keystone to understand the behavior of caching algorithms along any routing path in a generic topology. We assume that each node is associated with a single user,

TABLE I. MAIN NOTATION

Symbol	Meaning
N	Number of network nodes/users
C	Number of different contents
B	Cache size in number of contents

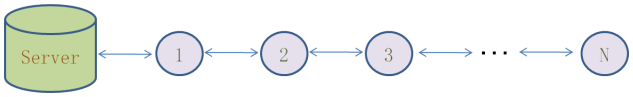

Fig. 1. Multi-hop linear topology with one server and N nodes. All the content requests travel towards the left and the contents travel towards the right.

who generates requests for different contents. The server is equipped with a finite catalog of C different contents, which are the only ones that can be requested by the users. Each node is equipped with a local cache, able to store up to B contents, independently from their size.

Whenever the user generates a request, the corresponding node sends a *request packet* (r_c) to request the content. We assume that the requests are always routed towards the server, i.e., to the left, as shown in Fig. 1. Each request is transmitted between neighboring nodes in a multi-hop fashion, until it reaches the first node having the requested content or the server. Then, the hit node (or the server) replies by sending back the *content* (c), which reaches the requester node in a multi-hop fashion. Note that in the case of distributed caching policies, the nodes along the path traversed by the content can eventually store it in their own cache. When the content reaches the requester node, the content is always locally cached.

The performance of a caching policy is evaluated in terms of the average distance traveled by a content request. The optimal caching policy minimizes such distance, thus it results in minimum content access latency and minimum number of transmissions, reducing network load and congestion.

III. RELATED WORK

Caching is an important technique to enhance the performance of wired and wireless networks. Caching eviction policies, as LRU (Least Recently Used) and some variants of it, have been widely studied [5], [6], [7]. Authors in [8] proposed GreedyDual algorithm which can be regarded as a generalization of LRU, as it tend to evict either "old" contents or contents with small access costs. On the other hand, LRU evicts just the oldest contents.

Interestingly, [9] proposes an analytical methodology to minimize the average distance to access a content in a large two-dimensional grid topology; as a result, the optimal caching policy must provide a number of copies proportional to $p^{0.667}$, where p is the request popularity of a content. Then through simulations, [9] shows that a local replacement algorithm, derived from GreedyDual in [10], is able to approximate the optimal solution. In our work, we adopted such policy, denoted as "CacheDistant" (see Algorithm 3 in Sec. IV).

Unlike eviction schemes, *insertion* policies are less studied. In the specific context of CCN [5], a universal caching approach is proposed, in which each node caches every new content; such policy has been referred as "1-DC" policy in our work. To reduce redundancy between contents cached in neighboring nodes, [11] suggested a group caching scheme in which only one node will cache the content among the one-hop neighbors. Thus, a node has to periodically update its neighbors about its cache status. Based on this information a node is selected for caching a newly arrived content.

To overcome the protocol overheads in the network when distributing a content across the network, a probabilistic caching scheme is introduced in [12]. The insertion decision is taken at random with a fixed probability $p \in [0.75, 0.9]$; this policy will be referred as "p-DC" policy in Sec. IV. Note that both 1-DC and p-DC require no information exchange between caches.

In the context of wireless ad-hoc networks, [13] (extending the work of [14]) considers a content dissemination scheme among the nodes based on the channels each user has subscribed. Whenever a node receives a content, if this belongs to a subscribed channel, it is stored in the node *private cache*. Otherwise, it is eventually stored in the node *public cache* that is present in the node to allow cooperative dissemination among the nodes. Differently from [13], we do not distinguish between private and public cache. In our proposed LC policy, the cache works as a private one, whereas in p-DC it works as public. Finally, we consider a fixed communication topology and we do not consider the effect of mobility.

Regarding the request patterns, recent works [15], [16] have suggested that geographic locality affects user-generated content consumption and content popularity. According to [16], about 50% of the videos have more than 70% of their views in a single region, showing the effect of geolocation on video on-demand. On the other hand, according to [17], about 71% of the contents are requested just once by a user, highlighting possible limitations of generic caching systems.

IV. CACHING POLICIES

We will investigate two families of caching policy, based on where the caching operations occur.

To define the caching algorithms, we define $d(c)$ as the distance of content c from the closest content provider (either a node or the server). Let $\mathcal{K}_n$ be the set of all the contents stored at a given node n; by construction, $|\mathcal{K}_n| \leq B$. Any content c present in cache of node n is associated with an eviction priority $H_u(c)$. Let $source(r_c)$ be the requester node for r_c.

Distributed Caching (p-DC) policy is a randomized algorithm with parameter $p \in [0, 1]$ and whose pseudocode is reported in Algorithm 1. The policy runs at each node n along the path from the content provider to the requester. This policy may cache a copy of the content along any node of the path, with a given probability p at each node. Also, the content is always cached at the requester node. Note that function *CacheDistant* in the pseudocode stores the content according to the eviction policy described later in this section and implements a variant of "least recently used" (LRU) replacement policy.

On the other hand, in *Local Caching (LC)* only the requester node caches content c, as shown in pseudocode of

Algorithm 1 p-DC (Distributed Caching)

Require: n is the current node, c is the content
  if $(c \notin \mathcal{K}_n)$ **then** ▷ c not in the local cache
    if (source$(r_c) == n$) **then** ▷ node is the requester
      CacheDistant(c)
    else ▷ another node requested c
      CacheDistant(c) with probability p
    end if
  else ▷ c in the local cache
    $H_n(c) = d(c)$ ▷ update priority with content distance
  end if
  forward c to the requester

Algorithm 2. Note that, by construction, 0-DC degenerates into LC.

Algorithm 2 LC (Local Caching)

Require: n is the current node, c is the content
  if $(c \notin \mathcal{K}_n)$ and (source$(r_c) == n$) **then**
      ▷ c not in the local cache and node is the requester
    CacheDistant(c)
  else ▷ c in the local cache
    $H_n(c) = d(c)$ ▷ update priority with content distance
  end if
  forward c to the requester

Furthermore, upon the arrival of a new content, if the cache is already full, the CacheDistant policy stores the new content after operating the eviction policy. This policy extends LRU, since it tries to evict either "old" contents or those with small distance from the source. Indeed, if another copy of the content is available close to the node, this can be evicted locally with high priority. Conversely, CacheDistant tends to keep within the cache the farthest contents, and this fact should explain the policy name. We have chosen CacheDistant since it has been shown to outperform LRU in networks of caches, as discussed in Sec. III.

In more details, CacheDistant keeps an eviction priority value $H(c)$ for every cached content $c \in \mathcal{K}$. A lower value of $H(c)$ means an higher priority to be evicted. When a new content is cached, its value is set equal to the distance $d(c)$ from the closest copy. In the case of eviction, the content $\hat{c}$ with the minimum value of H is removed and, finally, all the values of the remaining contents are decreased by $H(\hat{c})$. In this way, an aging mechanism is implemented to remove the oldest contents.

Algorithm 3 CacheDistant(c)

  if $(|\mathcal{K}| == B)$ **then** ▷ full cache
    $\hat{c} = \arg\min_{k \in \mathcal{K}} H(k)$ ▷ find the content to evict
    $\mathcal{K} = \mathcal{K} \setminus \hat{c}$ ▷ remove $\hat{c}$
    for each $k \in \mathcal{K}$ **do**
      $H(k) = H(k) - H(\hat{c})$ ▷ update all priorities
    end for
  end if
  $\mathcal{K} = \mathcal{K} \cup c$ ▷ store c into the cache
  $H(c) = d(c)$ ▷ update content priority

V. Methodology

We developed a Montecarlo simulator, written in C++, that models the linear topology (shown in Fig. 1), the request process at each node and the different caching policies under investigation. The simulator evaluates the performance in terms of distance to reach the closest copy, averaged across all content requests.

To compare the performance of different caching policies, we varied the size of the network N, the catalog size C and the cache size B. The simulation runs in discrete time steps. At each timeslot, a random permutation of the users is generated and, based on it, each user is sequentially chosen to generate a request for a content. The requested content is selected at random within the catalog, according to one of the following request scenarios: uniform, zipf, zipf-one-requests and trace-driven.

Under *uniform requests*, each user independently requests a content at random with uniform probability, i.e. all the contents have the same popularity. Note that a user can request the same content many times. This model is very simple, due to the limited level of redundancy in the request process, and even if unrealistic it aims at testing the caching performance under "worst-case" scenario.

Under *zipf requests*, the popularity of each content is assumed to follow a Zipf distribution with parameter $\alpha > 0$. More precisely, if q_k is the probability of requesting the kth content in the catalog, it holds:

$$q_k = \gamma / k^{\alpha} \qquad \text{with } k = 1, \ldots, C \tag{1}$$

where γ is a proper normalization factor. This traffic model is well-known, and captures the different content popularity in large content catalogs, as requested by an aggregation of users. Due to high level of redundancy in the request process, this process favors cooperative cache schemes. The zipf model is valid at an aggregated level, and may not be fully realistic when applied to each single user.

A. Zipf-one-requests

The last observation regarding zipf requests motivates a new request process, denoted as *zipf-one-requests*. Under this scenario, the global popularity of contents is still assumed to follow a zipf distribution, but now the popularity experienced by each user is not anymore a zipf, since each user is allowed to request each content at most once. To satisfy both features, we propose the following methodology to generate the request process.

Given the global zipf popularity according to (1), we impose that the kth content must receive exactly Nq_k/q_1 requests. Thus, the most popular content receives exactly N requests (one from each user) and all the other contents a number of requests proportional to its popularity, relatively to q_1. The request generator associates a given number of requests for a content to a random permutation of users, thus each content is requested at most once by each user. Each request is also associated to a random time. Finally, after all requests have been associated to a user and to a time, they are scheduled by the request generator of the simulator in increasing time order. Following this approach, the total number of generated

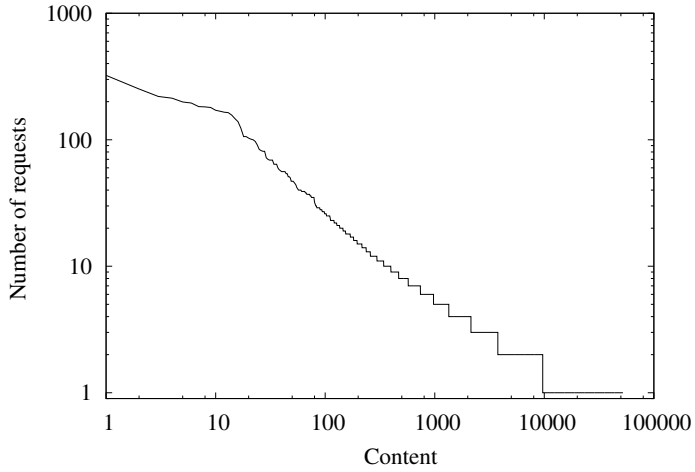

Fig. 2. Global popularity of the contents in the trace.

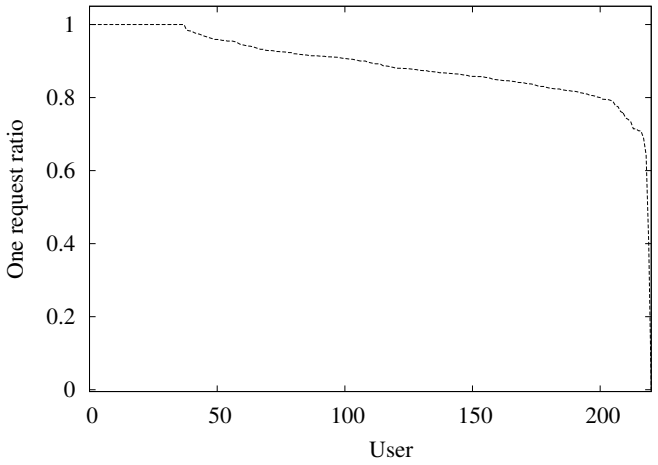

Fig. 3. One-request ratio for each user, reported in decreasing order.

requests is, by construction, equal to N/q_1. For example, for $\alpha = 0$ (i.e. uniform requests), the total requests are NC, and for $\alpha = 1$ the total requests are about $N \log C$.

Zipf-one-request model allows to consider the extreme case in which the request process at each user has null redundancy, even if it is still redundant at global level. We expect this scenario behaving as worst-case for local caching, since it has no particular value for a user to store its own requested content when he will never request it again.

B. Trace-driven requests

To evaluate a real scenario, we considered a YouTube traffic trace that was captured by one of the largest Italian ISP on a /24 IP subnet, which corresponds to a set of households attached to the same DSLAM at the central office, i.e. households in the same neighborhood. The trace collects the sequence of all YouTube videos downloaded by 220 distinct households during the period between May 1st and May 28th, 2012. The total number of downloads is 79,725, corresponding to $C = 52,133$ distinct videos. In our work, we mapped each household to a specific user in the linear topology, through a random permutation.

Fig. 2 shows the global popularity of the contents in the trace, which can be approximated by a zipf law with $\alpha = 0.7$. Moreover, out of all C contents only 20% have been requested more than once, while the remaining have been accessed just once in the whole trace. To show that this scenario could benefit from a caching system, consider that the fraction of requests for contents receiving more than one request is large (around 47%); furthermore, the cache hit probability would be 35% in the case all the requests were concentrated on a single cache, enough large to store all the contents.

We now consider each user's perspective and show in Fig. 3 the one-request ratio for each user. The one-request ratio is defined as the ratio between the number of contents requested just once by a user and the total number of requested contents by the same. Averaging across all users, 88.9% of contents are requested just once by a user. As extreme case, 47 out of 220

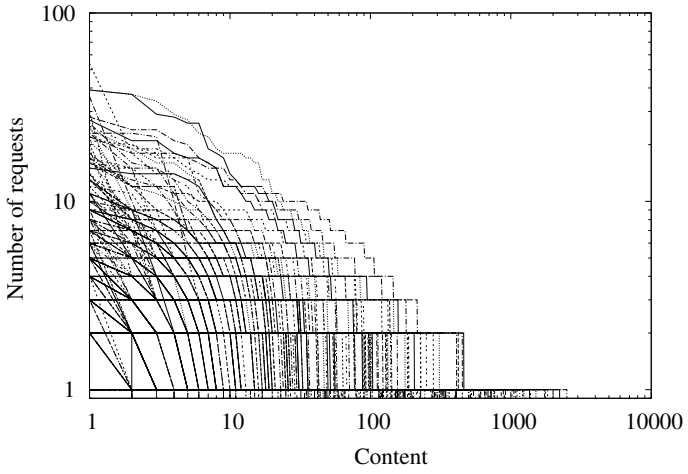

Fig. 4. Individual ranking of contents for users in the trace

users request a particular content just once in the whole trace. These observations show that it is not possible to apply the zipf model for such users and motivate the adopted zipf-one-request model. As other extreme case, the one-request ratio for one user is zero since all contents have been requested more than once.

Fig. 4 reports the popularity of the contents seen by each user, so in total 220 curves are shown. For each user, the contents have been sorted in decreasing popularity, independently from other users. From the picture it is clear that for some users and for a limited set of contents, the zipf model is still a good approximation, even if it cannot be applied to the whole set of users and contents, as discussed in the previous paragraph. From Fig. 4, it is also clear that some extreme cases cannot be approximated at all by a zipf model. In particular, some users requested around 3,000 distinct contents, out of a total of 52,133, which implies around 100 new contents for each day. For these users, a uniform request pattern would be a more appropriate model.

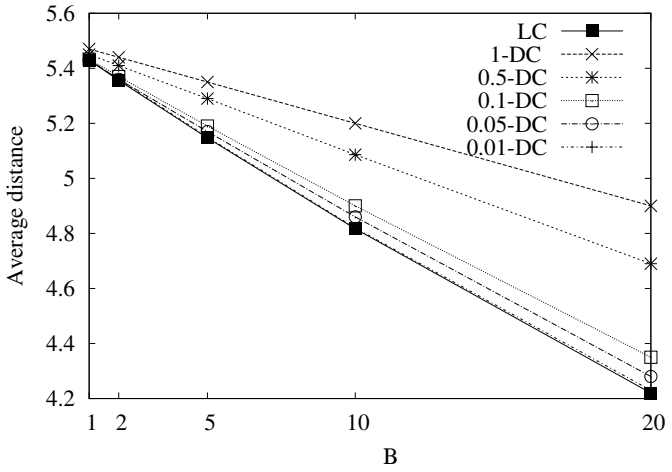

Fig. 5. Average distance for different caching policies under uniform requests, for $N = 10$ and $C = 300$.

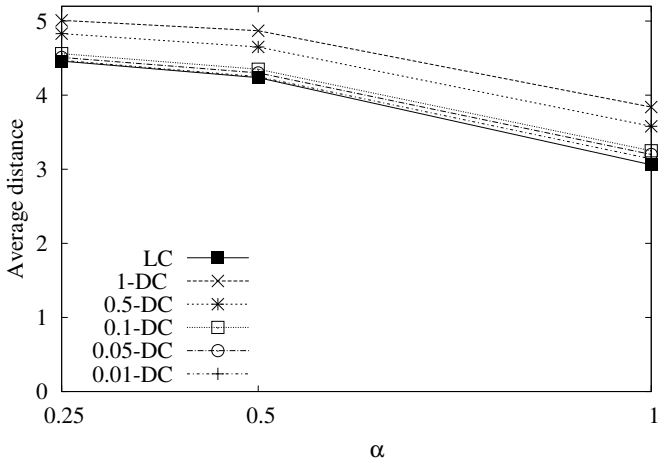

Fig. 6. Average distance for caching policies under zipf requests with parameter α, for $N = 10$, $C = 100$ and $B = 5$.

As a conclusion, reality cannot be described by a simple request model at single user level, and none of the presented synthetic models is perfect. Indeed, each of them can be considered meaningful only for a subset of users and contents, and all of them are useful to compare the performance of different caching systems.

VI. Simulation results

Fig. 5 compares the average distance for different caching policies, under uniform requests. We set $N = 10$ nodes and a catalog of $C = 300$ contents. We varied the local cache sizes B. We run the simulations for 10^5 timeslots, i.e. a total of 10^6 requests were generated. From the figure, it is clear that LC shows the best performance in terms of average distance. Indeed, under uniform requests the probability that two or more consecutive requests for the same content occurs is very low and it is better just to keep few copies of each content in the whole network. For each request of a new content, LC stores one copy, and it takes around BN requests to saturate the whole network buffer (defined as union of the nodes caches). Hence, we can expect $O(BN)$ distinct contents stored in the network buffer at the same time, and thus the hit probability, referred to the network buffer, for a generic user will be $O(BN/C)$. Instead, p-DC stores multiple copies of any new content, on average a number of copies equal to $pN/2$. In the worst case, 1-DC stores one copy for each node, i.e. around B requests are enough to saturate the network buffer, and thus the hit probability will be $O(B/C)$, much smaller than LC. Thus, LC consistently outperforms DC under uniform scenario.

Fig. 6 shows the average distance for zipf requests, for different values of α. We considered a network with $N = 10$ nodes, $C = 100$ contents and each cache of size $B = 5$ contents. The total simulation time was 10^5 timeslots and the number of requests equal to 10^6. The figure shows that, also in this scenario, LC policy outperforms the other policies and 1-DC has the worst performance. For larger values of α, the redundancy of requests increases and we can expect that it is better to store few popular contents everywhere in the network; indeed, the performance of DC improves compared to LC.

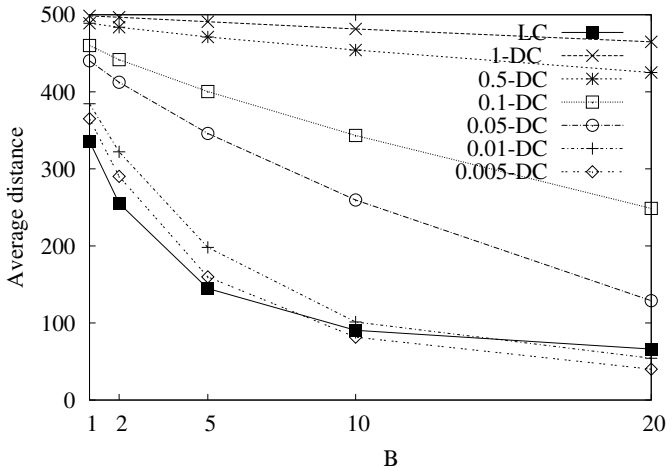

Fig. 7. Average distance for different caching policies under zipf-one-requests with $\alpha = 0.5$ for $N = 1000$ and $C = 1000$.

Note that, under zipf requests, both LC and DC tend to devote the caches to store just the most popular contents.

For the sake of space, we do not report the results for larger cache ($B = 20$). In this scenario, the performance of all the caching policies improves, since the caching becomes more effective. The same qualitative behavior is observed as the one obtained for $B = 5$, but now the effect of the different policies tends to vanish, since for enough large B all the policies behave almost the same.

Fig. 7 considers a larger network ($N = 1000$) under zipf-one-request traffic described in Sec. IV. We set $C = 1000$ contents and $\alpha = 0.5$ for the zipf global popularity; the requests were 61,728 and satisfy the constraint that a user does not request the same content more than once. When the cache is very small ($B = 1$), LC outperforms the other policies, with a relevant performance gain with respect to 1-DC policy. For large values of B, the beneficial effect of the cache size

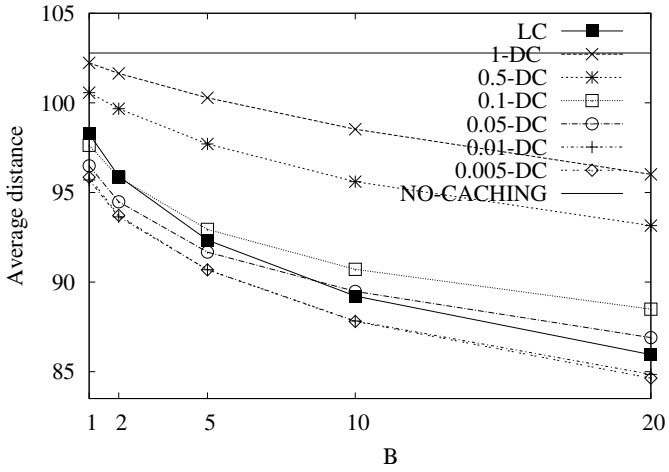

Fig. 8. Average distance different caching policies under trace-driven requests, havving $N = 220$ and $C = 52133$.

tends to vanish since the number of requests for each node is not enough large to fill the cache. Thus, LC is not able to store a variety of contents as in 0.01-DC and 0.005-DC, and the performance becomes slightly worst than them. We can conclude that LC is usually outperforming DC, even if in some corner cases a well-tuned value of p allows DC to slightly outperform LC.

Fig. 8 shows the average distance under trace-driven requests, discussed in Sec. V-B, for one random mapping between the households and the nodes in the linear topology. We recall that the total number of requests in the trace is 79,725. As a reference, we also show the performance achievable by NO-CACHING policy, which does not cache at all any content along the path. The corresponding average distance (equal to 102.8) depends on the specific household-node mapping and on the number of requests for each node. For any cache size B, 1-DC shows the worst performance among all caching policies, corroborating our previous findings. Instead, in this scenario LC is outperformed by 0.05-DC, 0.01-DC and 0.005-DC. The reason is that the number of requests is very variable among the nodes (from a minimum of 1 to a maximum of 3826 requests per node) and for the nodes with few requests the caches remains almost empty in LC. Thus, the cooperative effect of LC vanishes. Note that this effect, even if relevant for this trace, is mainly due to the limited size of the trace. Nevertheless, LC is behaving worse but comparable with DC, provided that p has been carefully tuned for the specific topology and request pattern. So, we can still conclude that LC is an efficient caching scheme due to the limited loss of performance and the simplicity of the approach, that does not require any parameter to be tuned to the running scenario.

VII. Conclusions

We have considered a network of caches implemented through D2D communications between neighboring mobile nodes. The network is organized according to a linear topology, in which each user corresponds to a node and cooperates with the other nodes enabling two main functionalities: infrastructure-less multihop communications and cooperative caching.

In such specific scenario, we have focused on cooperative caching and we have mainly investigated *where* to cache the contents, according to one of two different policies: under distributed caching policies, the content is eventually cached along all the nodes located between the requester node and the closest copy of the content. On the contrary, under a local caching policy, the content is cached only at the requester node. We compare the two policies under different request scenarios (also taken from real requests of YouTube movies measured on a large ISP) and show that a local policy, despite its simplicity, is usually outperforming distributed policies and, only in some corner cases, a local policy is slightly worse than a well-tuned distributed policy. These results advocate the implementation of local policies, but also question the adoption of more complex distributed policies.

References

[1] Wi-Fi Direct. [Online]. Available: http://www.wi-fi.org

[2] AllJoyn. [Online]. Available: http://www.alljoyn.org

[3] 3GPP release 12. [Online]. Available: http://www.3gpp.org

[4] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *IEEE Communications Magazine*, vol. 50, no. 7, pp. 26–36, July 2012.

[5] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *ACM Co-NEXT*, 2009, pp. 1–12.

[6] Y. Abdelmalek and T. Saadawi, "Collaborative multimedia content caching algorithms for mobile ad-hoc networks," in *IEEE MILCOM*, 2009, pp. 1–7.

[7] H. Gomaa, G. Messier, R. Davies, and C. Williamson, "Media caching support for mobile transit clients," in *IEEE WIMOB*, 2009, pp. 79–84.

[8] N. Young, "The k-server dual and loose competitiveness for paging," *Springer Algorithmica*, pp. 525–541, 1994.

[9] S. Jin and L. Wang, "Content and service replication strategies in multi-hop wireless mesh networks," in *ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, 2005, pp. 79–86.

[10] P. Cao and S. Irani, "Cost-aware WWW proxy caching algorithms." in *Usenix symposium on internet technologies and systems*, 1997, pp. 193–206.

[11] Y.-W. Ting and Y.-K. Chang, "A novel cooperative caching scheme for wireless ad hoc networks: Groupcaching," in *IEEE International Conference on Networking, Architecture, and Storage*, 2007, pp. 62–68.

[12] S. Arianfar, P. Nikander, and J. Ott, "On content-centric router design and implications," in *ACM Proceedings of the Re-Architecting the Internet Workshop*, 2010.

[13] L. Hu, J.-Y. Le Boudec, and M. Vojnoviae, "Optimal channel choice for collaborative ad-hoc dissemination," in *IEEE INFOCOM*, 2010, pp. 1–9.

[14] V. Lenders, G. Karlsson, and M. May, "Wireless ad-hoc podcasting," in *IEEE Communication Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, 2007, pp. 273–283.

[15] A. Brodersen, S. Scellato, and M. Wattenhofer, "YouTube around the world: Geographic popularity of videos," in *ACM 21st International Conference on World Wide Web*, 2012, pp. 241–250.

[16] Z. Li, G. Xie, J. Lin, Y. Jin, D. Kaafar, and K. Salamatian, "On the geographic patterns of a large-scale mobile video-on-demand system," in *IEEE INFOCOM*, 2014.

[17] M. Busari and C. Williamson, "ProWGen: a synthetic workload generation tool for simulation evaluation of web proxy caches," *Elsevier Computer Networks*, 2002.