A bloom-filter-based socially aware scheme for content replication in mobile ad hoc networks

Ghada Moualla, Pantelis A. Frangoudis, Yassine Hadjadj-Aoul, and Soraya Ait-Chellouche
IRISA-University of Rennes 1, Campus de Beaulieu, 35042 Rennes, France
emails: firstname.lastname@irisa.fr

***Abstract*—The volume of mobile multimedia traffic is fast-growing, challenging the radio and backhaul network infrastructure and calling for alternative content dissemination schemes. To improve user experience and reduce infrastructure load, we exploit implicit social relationships among users and take into account content popularity, proposing push-based prefetching mechanisms which take advantage of the caching and mobile ad hoc networking capabilities of user devices. We use Bloom Filters as summaries of user caches, and design mechanisms to estimate the social distance between users and the popularity of content items, which drive our algorithms. Our simulation-based evaluation shows that our scheme brings caching performance improvements in an order of 10% in terms of absolute cache hit ratio in most of the cases studied, and from 3% to 82% in terms of normalized cache hit ratio gain.**

I. Introduction

The spread of powerful and affordable mobile devices, such as smartphones and tablets, with multiple wireless interfaces such as 3G/4G/LTE and Wi-Fi, has resulted in a significant growth of mobile data traffic [1]. This increase overwhelms the capacity of wireless links, and decreases mobile data revenue. The gap between the huge traffic demands and the link (but also backhaul) capacity, along with link fluctuation conditions, can lead to poor Quality of Experience (QoE) for mobile users, such as start up delays and sporadic disruptions.

To address these problems, caching techniques are being explored [2], taking advantage of the storage capacity and the capabilities for direct local connectivity of user devices: By caching data at user terminals, a request can be served directly over device-to-device links, without being propagated to the data source each time, thus reducing mobile network infrastructure load. A critical issue is then to design efficient cache management strategies to make the best of the available storage, aiming to increase the number of cache hits. To this end, appropriate cache replacement policies and smart selection of content to proactively download to user caches (*prefetching*) should be in place. In this work, we focus on the latter, targeting a mobile ad hoc network (MANET) setting.

What and when to proactively cache is a thorny question, inherently involving a prediction process. Intuition, but also recent results [3], [4], indicate that the social aspect critically affects content access patterns, in the sense that users who are close in the social space are more likely to request similar content. This observation can drive efficient socially aware prefetching mechanisms, the focus of this work. There are various challenges to face to this end. A social proximity metric should be defined and appropriate mechanisms to measure it should be in place. In a MANET context, these mechanisms should be low-overhead and simple to implement. Our approach is to infer social relationships on the fly and in a distributed manner, solely based on user cache contents, instead of relying on external information (data from user past encounters, activities in existing social networks, etc.). Our approach is suitable for mobile Content-Centric Networks (CCNs), where nodes inherently support caching.

In this paper we make the following contributions: Using bloom filters [5] to summarize cache contents, we design distributed mechanisms to estimate the social distance between nearby users who form a MANET. We then propose a socially aware prefetching scheme, where users proactively push popular content to the caches of their peers, showing it to outperform no-prefetching or social/popularity-unaware caching approaches.

The remainder of this paper is organized as follows. In Section II we provide a short survey of the related literature. Section III presents our target environment and our proposed algorithms, for which we present simulation-based performance results in Section IV. In Section V we conclude the paper.

II. Related work

The emergence of Information/Content-centric network (ICN/CCN) architectures [6], [7] has brought research focus back to caching issues. To a significant extent, ICN/CCN target at matching content requests with available content items, rather than at connecting hosts, inherently supporting caching at various points in the network. In this context, content popularity is a critical aspect to take into account, as shown by Rossini et al. [8]. In the same spirit, Bernardini et al. [9] have proposed a new cache replacement strategy named Most Popular Content (MPC). In this strategy, only items whose popularity (calculated locally by each node as the number of times a request for an item has been observed) exceeds a fixed threshold are considered for caching.

Cache management strategies aside, proactive mechanisms for storing content in caches before actually being requested are being considered. Such prefetching schemes need means to

The research leading to these results was performed under the CAMION project, which has received funding from Eurostars.

978-1-4673-9292-1/16/$31.00 ©2016 IEEE

accurately predict future requests, with the aim of improving response times and saving network resources by serving requests from local stores. As with cache management, prefetching can be driven by data popularity. Furthermore, exploiting the social aspect for predicting future content requests has started to be explored in various contexts. Bernardini et al. [10] showed that identifying influential social network users and preferring to prefetch the content they publish is advantageous in a CCN environment. Wang et al. [3] focus on a cloud-based scalable video delivery scheme for mobile users. The authors considered the type of social links and the specific activities of the users in the social network. For example, a video directly recommended to a user by a friend is more likely to get prefetched than a video published to everyone.

The distinctive characteristic of our approach compared to the aforementioned schemes is that we do not make any assumptions about the presence of users in social networks and their activities. We estimate social relationships indirectly, solely based on the similarity of the contents of user caches, which makes our system lightweight and generic. This similarity metric could be considered an instance of the "social proximity" abstraction introduced by Iqbal and Giaccone [4], who focus on socially-aware cache insertion for wireless CCNs. Indeed, they used the concept of social distance, defined as the euclidean distance in a generic multi-dimensional social space model.

We are using a cache representation similar to Summary Cache [11], a cache sharing protocol for Web proxies. In that work, each proxy maintains counting Bloom Filters which summarize every other proxy's contents. In the event of a local cache miss, the proxy will look up the requested item in these Bloom Filters and redirect the request to the appropriate proxy[1]. A similar approach for sharing content store representations is followed in a position paper by Kazi [12], aiming to control the flooding of interest packets in CCNs: A router capable of responding to an interest packet, also transmits a bloom filter representing all items resolvable by it, which traverses all CCN routers in the message path back to the requestor.

III. Architecture and Mechanisms for Proactive Content Replication

A. Architectural aspects

1) Targeted Environment: This work targets future content-centric mobile networks where traffic-intense multimedia applications dominate. These networks are characterized by scarce resources, at both the access and the back-haul levels, which aggravate the data delivery problem. It thus becomes harder for mobile network operators to satisfy user demand. In particular, we focus on the case where mobile users are located in the same physical space and form mobile ad-hoc networks, taking advantage of the communication capabilities of their terminal equipment.

[1]Since cache contents are changing frequently, a counter is maintained for each bit in the vector. If an element mapping to this bit is added/removed, the counter is incremented/decremented. If it reaches zero, the bit is also zeroed.

In this network, by exploiting device-to-device connectivity, the caching capabilities of these devices, and information from the social layer, optimized multimedia content delivery can be achieved, while reducing traffic loads on the network infrastructure.

2) Node Design: We assume a mobile CCN, where each node has the ability to cache a limited volume of content items. We represent cache contents using a Bloom Filter (BF) [5]. A BF is an n-bit vector, and k independent hash functions $(h_1, h_2, \ldots, h_k)$ are used to map an element to bits in the vector. To add an element e, all the bits at positions $h_1(e), h_2(e), \ldots, h_k(e)$ in the bit vector are set. When queried about the presence of an element, based on the vector positions pointed to by the hash functions, the BF responds either with "possibly in the set" or "definitely not in set." The more the elements added to the set, the higher the probability of a *false positive*. Selecting the appropriate BF parameters involves a tradeoff between space (BF size), accuracy (false positive rate) and processing (required hash function operations): For a given set of elements, appropriately selecting the size of the bit vector and the number of hash functions to use can guarantee a specific maximum false positive rate. For example, it can be shown that 8 bits per element and three hash functions can guarantee a false positive rate of approximately 3%.

The mechanisms we propose assume a level of awareness of the content accessed by peers. Therefore, each node needs to have some information about the cache contents of other nodes. To reduce message traffic, instead of transferring detailed information on the exact cache contents (e.g., URIs of content identifiers), nodes periodically broadcast BFs that summarize the contents of their caches. The advantage is space efficiency, since a BF needs only a fixed number of bits per element.

In our node model, there are two basic processes taking place:

1) *The foreground operation*, which encompasses content requests generated by direct user activities (for example, a user requests to view a video, a running application requests a content item, etc.).
2) *The background operation*, where prefetching activities take place. Our prefetching mechanism aims to predict content requests that could be carried out by the foreground operation, in order to reduce content access delay and improve user experience.

In the next section, we present the background operation in detail.

B. A Socially-Aware Content Delivery Mechanism

Our scheme builds on the premise that users who store similar contents are likely to have roughly the same preferences and are expected to request common content items in the future. Therefore, we consider the similarity among cache contents as an expression of implicit social proximity.

Exploiting this social aspect can enhance caching operations and bring benefits both to users and network operators, by (1) proactively caching content based on the expectation that

a user will request specific content items given his interests and the actions of socially proximate peers (this does not require physical proximity) and (2) utilizing device-to-device communication for faster lookup and delivery from the caches of nearby "friends."

Estimating social proximity between users is a significant challenge. Its accuracy depends on the level of information available. Opting for a generic and lightweight mechanism, instead of resorting to detailed user profiling or exploiting exhaustive information from social networking services, we take advantage of our Bloom-Filter-based design. We define the social proximity of user A in relation to user B as the ratio of content items in A's cache which can also be found in B's cache (see algorithm (1) for more details).

Algorithm 1 Selection of the best similar cache

Require: P ▷ Set of peers
Require: C ▷ Local cache

```
1: function GETBESTPEER(P, C)
2:     for all p ∈ P do
3:         SS[p] ← 0                          ▷ Similarity score for p
4:         for all f ∈ C do
5:             if (CHECKBF(p, f) = True) then
6:                 SS[p] ← SS[p] + 1
7:             end if
8:         end for
9:     end for
10:    return MAX(SS)   ▷ Index of the most similar cache
11: end function
```

The use of bloom filters as summaries of user caches facilitates calculating this metric. Instead of exchanging the full list of cached items, users exchange the respective BFs, which summarize them. This allows a particular user to look up its items, using the "CheckBF" function, in the BFs of other nodes, which enables determining social proximity.

The proposed approach has the following advantages:

- *Bandwidth savings:* Every node must know other nodes' content. By utilizing a BF-based cache representation, only a few bits are necessary per element. As a consequence, every node can send to the other nodes a small bit string instead of a detailed and exact representation of its cache, reducing the signalling overhead and bandwidth consumption.
- *Simplicity:* Contrary to mechanisms that depend on user profiling [4], [13], our approach is only based on user cache contents.
- *Privacy enhancements:* By exchanging BF-based cache summaries instead of the explicit cache contents, a level of privacy can be offered [14]. A thorough study of this aspect and quantifiable privacy advantages and potential threats are outside the scope of this work.

In our approach, apart from the social relationship between users, as estimated by the similarity function described above, we also consider content popularity. Assuming that content request patterns are not uniform, but rather, follow a Zipf distribution[2], we propose that the selection of which content to proactively cache should be biased towards popular items. Thus, after selecting a peer as a "friend" (i.e., a user with a high similarity metric), rather than a choice uniformly at random, we consider the most popular file for caching (not already existing in the destination cache).

Instead of having a user decide on the item to prefetch in his own local cache, we opt for a *push* approach for proactive caching. After selecting a socially close peer, a user decides to place a content item in the latter's content store. This design choice is guided from practical reasons: A user is only aware of his own cache contents and of the summaries (BFs) of peer caches, therefore he cannot explicitly pull a specific item from a peer. However, he can look up an item from his own cache in the peer's BF and, if lookup fails (BFs allow for no false negatives), to push the item there. Note that this approach requires a level of cooperation and trust among peers; in this work, we do not consider attackers pushing malicious content to peer caches.

The above description implies that we need a scheme to measure content popularity. Our procedure to estimate it is as follows: For each file in his cache, a user looks it up in all the BFs received in the similarity estimation process, and updates item occurrence counters according to algorithm 2.

Algorithm 2 Calculate file popularities

```
1: function GETFILESPOPULARITY(P, C)
2:     for all f ∈ C do  ▷ Browse all the files of the cache
3:         S(f) ← 0                          ▷ Score for the file f
4:         for all p ∈ P do
5:             if (CHECKBF(p, f) = True) then
6:                 S(f) ← S(f) + 1
7:             end if
8:         end for
9:     end for
10:    return S                              ▷ File popularities
11: end function
```

The push-based prefetching procedure, including the peer selection and popularity calculation steps, which is executed periodically, is summarized in algorithm 3. Once a file is prefetched, a local caching decision needs to be performed. In this work, and in the evaluation that follows, we are applying the Least-Recently-Used (LRU) replacement policy, but other options are also possible.

IV. Performance Evaluation

A. Simulation settings

In this section, we present a simulation-based performance evaluation of our scheme based on a custom discrete event-driven simulator we developed in C.

[2]There is empirical evidence indicating such a trend, and a significant body of relevant research follows this assumption [15], [16].

Algorithm 3 Bloom-Filter-based Prefetching Algorithm (BFPA)

Require: T ▷ Period of time

```
1: procedure BFPA(P,C)
2:     RECEIVEBFS()                ▷ Receive BF from others
3:     SENDBF(C)                   ▷ Broadcast its BF to others
4:     prefetched ← False
5:     S ← GETFILESPOPULARITY(P, C)
6:     N ← GETBESTPEER(P, C)
7:     repeat
8:         f ← MAXBF(S)            ▷ Return the most popular
9:         if (CHECKBF(N, f) = False) then
10:            PREFETCH(N, f)      ▷ Prefetch (push) file f to peer N
11:            prefetched ← True
12:        else
13:            S ← S − {f}
14:        end if
15:    until (prefetched or (S = ∅))
16: end procedure                  ▷ Prefetching procedure
```

Our performance metric of interest is the total *cache hit ratio*, representing the number of requested files satisfied by local caches divided by the total number of requested files.

Our simulator was configured to generate requests for content following a Zipf-like distribution, where the probability that the i-th most popular content item is requested is given by $q(i) = \frac{1}{i^{\alpha}}$, where α represents a popularity parameter (i.e. Zipf parameter).

Different values of α were considered (Table I), according to the scenarios described by Fricker et al. [15], [16], derived from real traces.

Internet content trac	Popularity (α)
Web	0.8
File sharing	0.8
VoD	1.2
User-generated content (UGC)	0.8

TABLE I
INTERNET CONTENT TRAFFIC CHARACTERISTICS (SOURCE [15])

The essential motivation for performing discrete-event simulation is to understand the effects of various parameters on the performance of our algorithm. We assume that there are different file objects with the same fixed size, and, given their Zipf-like distribution, the request frequency for the i-th object is $q(i)$.

To emulate different communities of users with common interests, we have assumed that the user population is evenly split in two classes, and different files are popular for each class, following the same Zipf law (i.e., same value for α, but mapping to different files).

The time interval between two successive requests generated from each node follows an exponential distribution with a mean value $mtba$. All nodes implement an LRU cache replacement policy. Note that prefetching starts after a period of time necessary to fill completely all the caches. The proposed algorithms are executed periodically as a background operation for every node.

In all our experiments, the simulation lasted 30000 time units. Unless otherwise noted, the mean time between arrivals was set to $mtba = 6.5$ time units, 10 users were participating in the ad hoc network, and each node was periodically carrying out the prefetching process every 30 time units.

The performance evaluation of our scheme is designed to study the effects of different content access patterns (α, time intervals between requests), file populations, and cache sizes on our main metric of interest, i.e., the cache hit ratio achieved. We compare 4 different strategies: (i) Our socially aware and content popularity-based selection mechanism (social+popular), (ii) a scheme where we ignore content popularity and instead select to push a random file to a socially proximate peer (social+random), (iii) a mechanism where a random item is pushed to a random peer (random+random), and (iv) the case where prefetching is disabled (no-prefetching).

We present the improvements in terms of cache hit ratio both as absolute values, and as the *normalized gain* compared to the hit ratio achieved by the baseline no-prefetching scheme (given by $\frac{HR_p - HR_0}{HR_0}$, where HR_p and HR_0 are the hit ratios for our scheme and the baseline one, respectively). Each reported result is the mean value of a few tens of iterations for the same parameter set, presented with 95% confidence intervals.

In all our experiments, we used a BF with three hash functions, and used 8 bits per set element, which limits the false positive rate to approximately 3%. For example, in an experiment where the total file population is 1000, each BF is 8 kb.

B. Experimental results

1) Overview of results: We found that, with socially aware and popularity-based prefetching, we can always achieve a higher average hit ratio compared to the competing strategies. The improvement compared to the case where prefeching was disabled was in the order of 10% in absolute cache hit ratio values in most of the simulated settings, and the normalized gain compared to the no-prefetching strategy ranged from 3% (in scenarios where even plain LRU caching reaches hit ratios close to 1, e.g., when α is very high; see Fig. 1) to 82% (when prefetching takes place frequently compared to the request rate and $\alpha = 0.8$; see Fig. 4).

Such an increase in the amount of requests served from local caches can translate to a reduction in bandwidth utilization: For example, for a repository of 1000 files of 10 MB each, an average hit ratio increase of 5% (a conservative estimate, since in the majority of our tests the improvement was more than that) can bring an average reduction in bandwidth utilization in the order of 500 MB, depending on the topology.

We also confirmed that the relative cache size to the total file population positively affects performance for all four

candidate schemes, with our proposed mechanism consistently outperforming its candidates.

The Zipf parameter is also significant: The less uniform requests for content items are, the more significant considering content popularity becomes, as shown by the improved hit ratio achieved as α grows.

We should remark that the plain (no prefetching) caching strategy and the random prefetching one (random+random) perform almost identically, while our proposed prefetching scheme always outperforms the popularity-unaware one. This justifies the need to take content popularity into account. Finally, due to the lower absolute hit ratio values and the way the normalized gain is calculated, in scenarios where α is small, this metric has a higher value compared to cases with a higher Zipf distribution exponent.

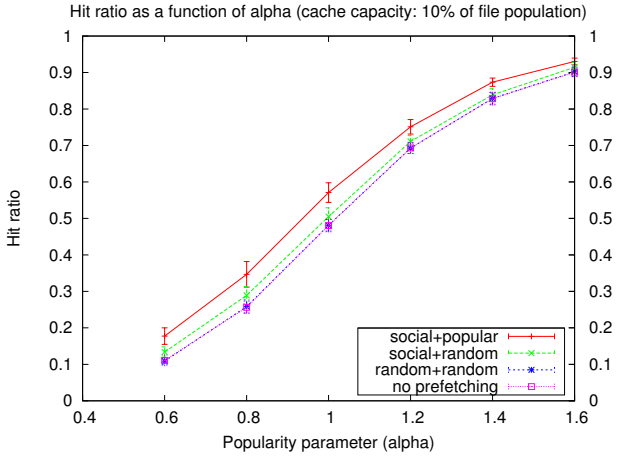


Fig. 1. Cache hit ratio as a function of the Zipf parameter for a fixed cache size to file population ratio. The total number of content items in the system is 10000, cache capacity is fixed to 1000, and there are 10 users.

2) Impact of content access patterns on performance: The Zipf parameter (α) determines the access pattern of mobile nodes. When it is small, the access distribution is more like a uniform one, hence there are a lot of files with similar popularity. In this sense, there is a high probability that most of the popular files are not in the cache when they are requested, thus the lower hit ratio. On the other hand, when α is large, requests are mainly towards the most popular content (frequently accessed data) and the number of highly popular files is small, so there is a higher probability that the requested files are already in the cache. As expected, this affects positively all candidate schemes.

Fig. 1 shows how different access patterns affect the hit ratio, for a fixed cache size to file population ratio. (As we shall further show in Section IV-B3, caching performance increases with cache size, keeping the file population fixed.) Our socially aware popularity-based mechanism outperforms all candidate strategies in all the cases studied. The same behavior is exhibited for other parameter settings. For the results presented in Fig. 1, we measure mean absolute hit ratio improvements of up to 9% (for $\alpha = 1$), while the normalized gain achieved compared to the no-prefetching strategy ranges from 3% to 60%.

3) Impact of cache size on performance: We study here the impact of the cache size on the cache hit ratio. First, we fix cache capacity to 500 items and vary the number of files in the system, for two different values of α. The results of this experiment are shown in Fig. 2.

As in all our experiments, smaller values for α come with smaller hit ratios for any candidate algorithm, but also with a higher normalized gain for our proposed scheme. In the experiments of Fig. 2, this metric ranges from 20% to 32% when $\alpha = 0.8$, while 9-11% normalized gains are achieved for $\alpha = 1.2$.

Then, we fix the file population to 10000 items, and vary the capacity of user caches. Fig. 3 shows the evolution of the cache hit ratio as cache capacity increases from 2% to 10% of the total number of items in the system, for two different content access patterns.

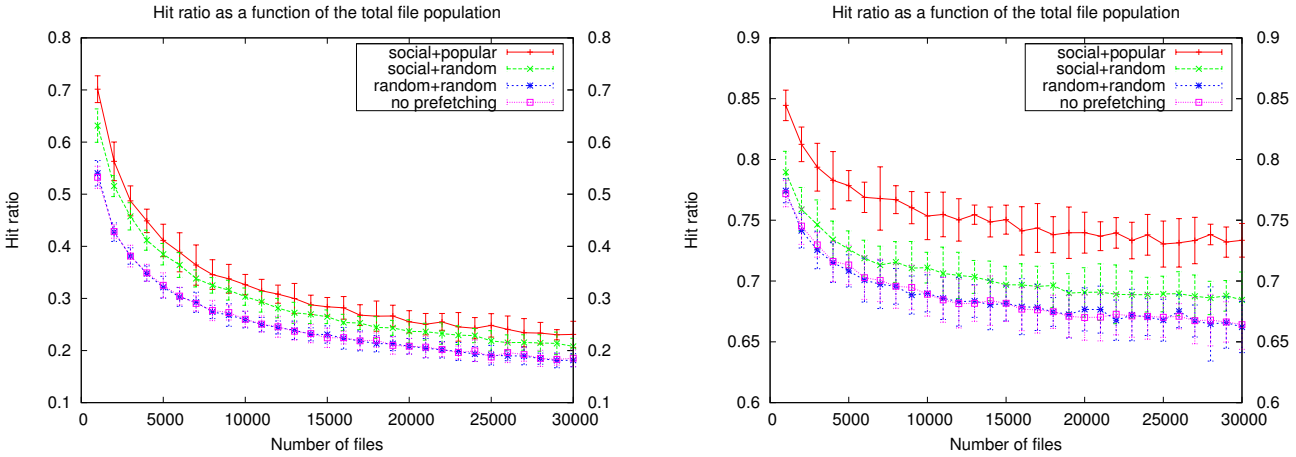
As shown in Fig. 3, the total hit ratio of our prefetching algorithm is always better compared to the alternative techniques (up to 10% absolute and 37% normalized gains when $\alpha = 0.8$, and up to 8% absolute and 11% normalized gains when $\alpha = 1.2$).

We can further notice that when $\alpha = 0.8$, the hit ratio when prefetching is disabled (LRU-caching-only scheme) is smaller than the hit ratio when $\alpha = 1.2$. Indeed, when α is small, the number of highly popular files is larger, so increasing the cache size will permit to store more of them, which subsequently increases the cache hit ratio significantly. However, for very large cache sizes, the cache hit ratio stabilizes as almost all popular files are already stored, and the cache starts storing unpopular ones, which are less likely to be concerned by future requests (i.e., no effect on the cache hit ratio).

4) The role of content request and prefetching frequencies: Caching performance appears to be sensitive to the frequency of content requests. As shown in Fig. 4, as request frequency decreases (increased mean time between arrivals), the hit ratio decreases for all candidate strategies. In fact, when increasing the interval between request arrivals, the number of the popular files is decreased due to the replacement strategy in favor of unpopular files, which in turn diminishes the total hit ratio.

However, we note that, while the hit ratio has an overall decreasing trend, the performance improvement of our scheme compared to the other strategies persists or gets higher, for lower request frequencies (foreground operation) and keeping the prefetching period fixed to 30 time units. Compared to the baseline no-prefetching scheme, this leads to up to 6% absolute and 82% normalized gains when $\alpha = 0.8$, and up to 11% absolute and 26% normalized gains when $\alpha = 1.2$.

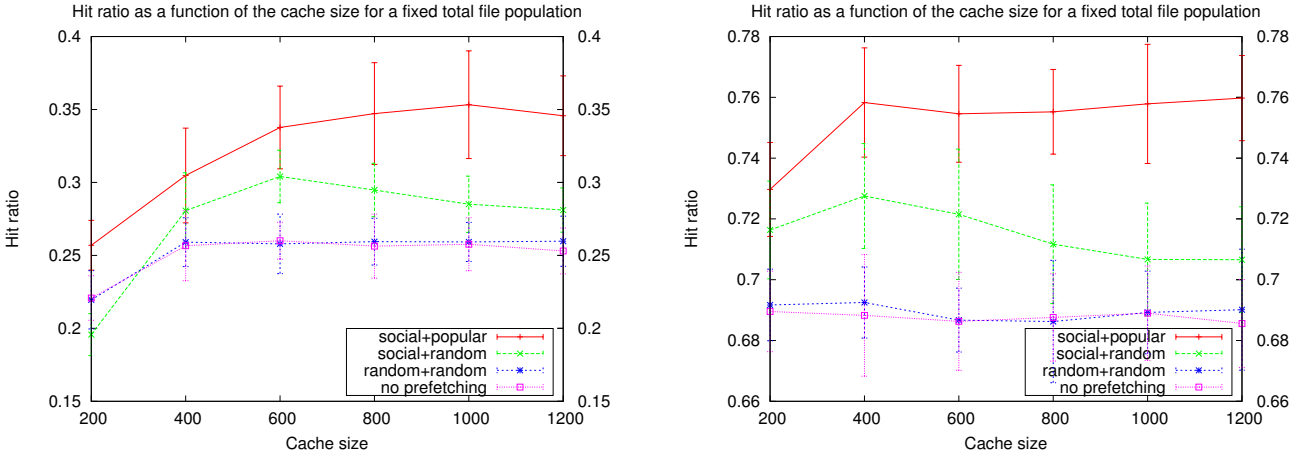
On the other hand, Fig. 5 presents the evolution of caching performance as prefetching frequency drops (i.e., the fixed interval between periodic background prefetching operations increases). Notably, a socially aware mechanism which does not take into account content popularity (social+random curve), while being competitive when prefetching takes place aggressively, eventually reduces performance-wise to a no-



(a) Cache capacity: 500 items, $\alpha = 0.8$.

(b) Cache capacity: 500 items, $\alpha = 1.2$.

Fig. 2. Cache hit ratio as a function of the number of files in the system, for a fixed cache size (500 items), in a 10-user setup.



(a) $\alpha = 0.8$

(b) $\alpha = 1.2$

Fig. 3. Cache hit ratio as a function of the cache size, for different content access patterns. The number of files in the system is fixed to 10000 and there are 10 users.

prefetching or a random scheme, for smaller prefetching frequencies. This implies that our scheme has a type of robustness to prefetching frequency, which is advantageous, since it translates to fewer proactive content requests (thus bandwidth savings), for a similar caching performance level. The same behavior is observed for different content access patterns (values of α).

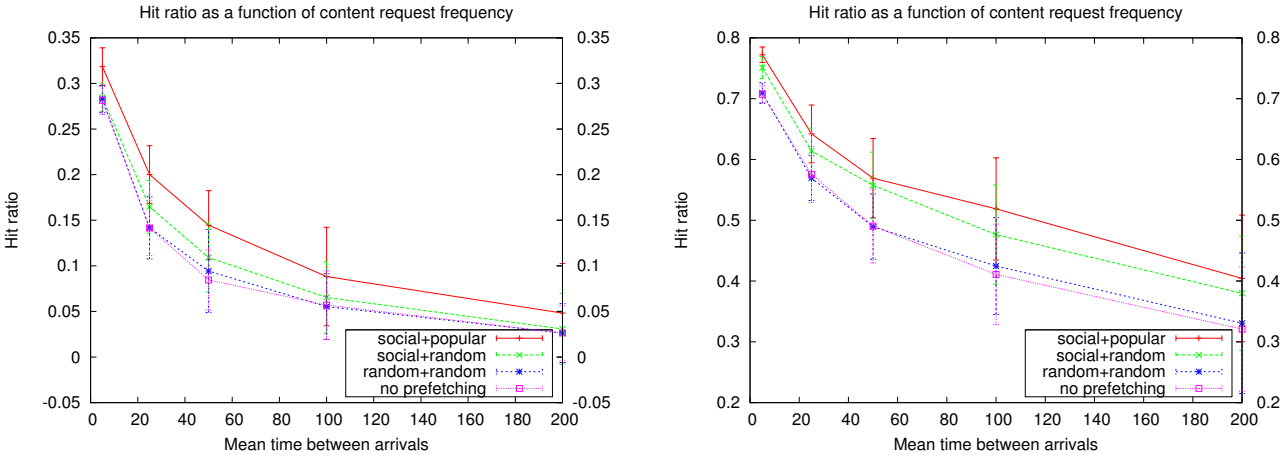
V. Conclusion and Future Work

In this paper, a push-based proactive content replication strategy has been proposed as a technique to enhance application performance in a mobile ad hoc network setting. We have shown that (i) exploiting the social proximity between users, indirectly estimated by the similarity of the content consumed by them, and (ii) information about content popularity, more content can be served from local caches, thus reducing startup delays. Building on a compact Bloom-Filter-based user cache representation, our distributed mechanisms can be implemented with reduced bandwidth demands.

Our future work will focus on performance enhancements and a more extensive evaluation of our scheme, both theoretically and via simulation, studying the effects of cache replacement strategies beyond LRU. At the same time, our design allows for integration in a content-centric networking software prototype. In another line of research, we are planning to study the privacy properties of our design, considering the security and privacy requirements of various application environments, and exploring alternative architectural solutions for secure and privacy-enhanced content search and replication.

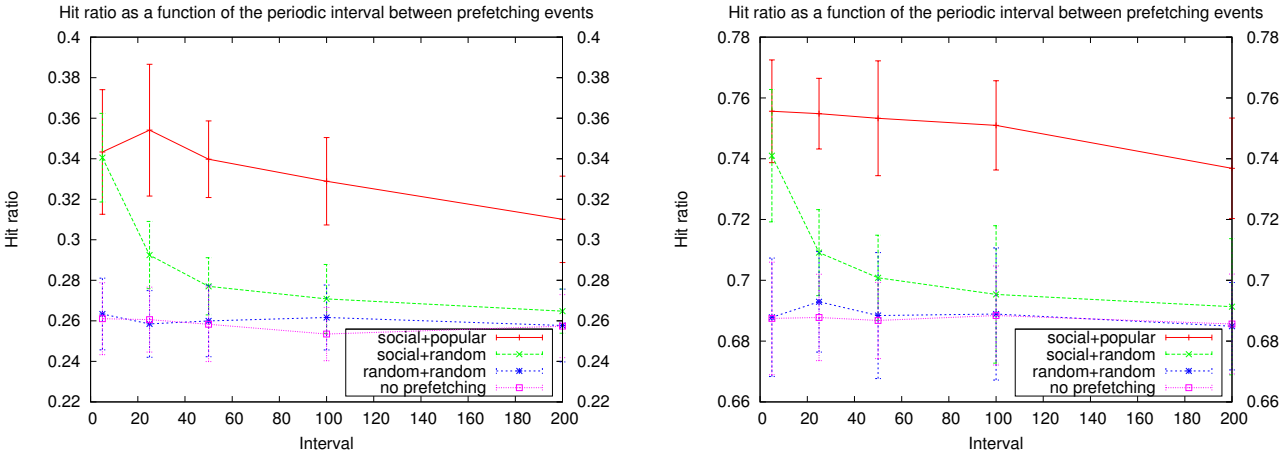
References

[1] Cisco, "Global mobile data traffic forecast update, 2013-2018," http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html/, 2014.



(a) $\alpha = 0.8$ (b) $\alpha = 1.2$

Fig. 4. Cache hit ratio as a function of the mean time between requests for a fixed prefetching period of 50 time units, in a setup of 10 users, 10000 files, and with the cache size fixed to 400 items.



(a) $\alpha = 0.8$ (b) $\alpha = 1.2$

Fig. 5. Cache hit ratio as a function of the time between two periodic prefetching events. The mtba is fixed to 6.5 time units, in a setup of 10 users, 10000 files, and a cache size of 400 items.

[2] A. Vakali and G. Pallis, "A study on web caching architectures and performance," *Proc. 5th World Multiconf. on Systemics, Cybernetics and Informatics (SCI 2001)*, 2001.

[3] X. Wang, T. T. Kwon, Y. Choi, H. Wang, and J. Liu, "Cloud-assisted adaptive video streaming and social-aware video prefetching for mobile users," *IEEE Wireless Communications*, vol. 20, no. 3, 2013.

[4] J. Iqbal and P. Giaccone, "Interest-based cooperative caching in multi-hop wireless networks," in *Proc. IEEE Globecom Workshops*, 2013.

[5] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[6] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proc. ACM CoNEXT*, 2009.

[7] G. Xylomenos, C. N. Ververidis, V. A. Siris, N. Fotiou, C. Tsilopoulos, X. Vasilakos, K. V. Katsaros, and G. C. Polyzos, "A survey of information-centric networking research," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 2, pp. 1024–1049, 2014.

[8] G. Rossini and D. Rossi, "A dive into the caching performance of content centric networking," in *Proc. IEEE CAMAD*, 2012.

[9] C. Bernardini, T. Silverston, and F. Olivier, "MPC: Popularity-based Caching Strategy for Content Centric Networks," in *Proc. IEEE ICC*, 2013.

[10] C. Bernardini, T. Silverston, and O. Festor, "Socially-aware caching strategy for content centric networking," in *Proc. IFIP Networking*, 2014.

[11] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM Transactions on Networking*, vol. 8, no. 3, pp. 281–293, 2000.

[12] A. W. Kazi, "Prefetching bloom filters to control flooding in content-centric networks," in *Proc. ACM CoNEXT Student Workshop*, 2010.

[13] N. Chauhan, L. Awasthi, and N. Chand, "Data caching with intelligent prefetching in mobile ad hoc networks," in *Proc. 2011 International Conference on Communication Systems and Network Technologies (CSNT)*, 2011.

[14] G. Bianchi, L. Bracciale, and P. Loreti, ""Better than nothing" privacy with bloom filters: To what extent?" in *Privacy in Statistical Databases*, ser. Lecture Notes in Computer Science, J. Domingo-Ferrer and I. Tinnirello, Eds. Springer, 2012, vol. 7556, pp. 348–363.

[15] C. Fricker, P. Robert, and J. Roberts, "A versatile and accurate approximation for LRU cache performance," in *Proc. 24th International Teletraffic Congress (ITC 24)*, 2012.

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