To Cache or Not To Cache?

Marco Fiore, Francesco Mininni, Claudio Casetti, Carla-Fabiana Chiasserini
Dipartimento di Elettronica
Politecnico di Torino
Torino, Italy
Email: {firstname.lastname}@polito.it

***Abstract*—We address cooperative caching in mobile ad hoc networks where information is exchanged in a peer-to-peer fashion among the network nodes. Our objective is to devise a fully-distributed caching strategy whereby nodes, independently of each other, decide whether to cache or not some content, and for how long. Each node takes this decision according to its perception of what nearby users may be storing in their caches and with the aim to differentiate its own cache content from the others'. We aptly named such algorithm "Hamlet". The result is the creation of a content diversity within the nodes neighborhood, so that a requesting user likely finds the desired information nearby. We simulate our caching algorithm in an urban scenario, featuring vehicular mobility, as well as in a mall scenario with pedestrians carrying mobile devices. Comparison with other caching schemes under different forwarding strategies confirms that Hamlet succeeds in creating the desired content diversity thus leading to a resource-efficient information access.**

I. Introduction

A vehicle is lumbering along in a typical morning rush-hour traffic jam. Its passengers are flipping through the information carried by an on-board display, looking for the latest news items, weather forecasts, traffic updates, or movie trailers. Such information content is provided by broadcast gateways run by transportation systems, local businesses, or else privately owned. It is carried around and exchanged by swarming vehicles equipped with networking devices in everyday city traffic, or by palmtops or smart phones belonging to people strolling on sidewalks. Although some may dismiss this scenario as too far off in the future, infotainment devices are being deployed on new cars, and an on-board and roadside network infrastructure is expected to follow along.

However, several fundamental issues are still being debated: what information distribution algorithm has the best chance to reach most mobile users; what caching strategy is most appropriate in an environment where a cache-all-you-see approach is unfeasible but where the availability of popular information from nearby nodes is the key to success.

In this paper, we provide an answer to the latter issue, i.e., how vehicles or pedestrians equipped with networking devices in an ad hoc network can efficiently cache information they receive *without swamping their limited storage capacity with needless information picked up on the go*.

Efficient caching in cooperative fashion and cache placement in wireless networks have been explored, among others, by [1], [2], [3], [4], [5]. In particular, the work in [1] proposes a cooperative caching scheme that however requires the nodes to periodically broadcast their identity as well as their cache contents. In [2], Yin and Cao present distributed caching strategies for ad hoc networks, according to which nodes may cache highly popular contents that pass by, or record the data path and use it to redirect future requests. We will take this proposal as term of comparison while evaluating the performance of our solution. The work in [3] presents both a centralized and a distributed solution to the cache placement problem of minimizing data access cost when network nodes have limited storage capacity. The distributed scheme, however, makes use of cache tables which, in mobile networks, need to be maintained in a similar vein as routing tables. Some approaches to eliminate information replicas among neighboring nodes are introduced in [4]. The schemes presented in [4], however, require knowledge of the information access frequency and periodic transmission of control messages that allow nodes to coordinate their caching decisions. In [5], the authors observe that to improve data accessibility, mobile nodes should cache different data items than their neighbors. In particular, the solution presented there aims at caching copies of the same content farther than a given number of hops, which again may be unsuitable for highly dynamic network topologies.

The solution we propose, called Hamlet, differs from previous work in that it helps users decide what information to keep, and *for how long*, based on a probabilistic estimate of what is cached in the neighborhood.

The objective is to create a *content diversity* within the node neighborhood, so that users likely find a copy of the different information items nearby and avoid flooding the network with query messages. Caching decisions are taken by each node separately, and considering only content query and information reply messages that a node "sees" on the channel. Thus, Hamlet is a fully distributed scheme and does not require the network nodes to exchange any additional control messages. The results we derive show that Hamlet is able to ensure a high query resolution ratio while maintaining the query traffic load very low, even for scarcely popular contents. Furthermore, when different copies of the same information are injected in the network and consistency becomes an issue [6], [7], Hamlet allows the network nodes caching the information to quickly replace the outdated content with its most recent version.

II. System Outline

In our vision, Hamlet can work with *any* content distribution system where a caching node can:

- overhear query/response messages;

- estimate its distance in hops from the query source node and the responding node;
- identify the version of the transiting information;
- associate a specific drop time to each item in its cache.

That said, we outline a system that fits the above requirements and that is instrumental to investigating the performance of Hamlet. The system we address is a mobile ad hoc network, where users may be either pedestrian (as within a mall) or vehicles (as on city roads). We assume that a number I of *information items* is available to the users, each item divided into C *chunks*, small enough so that each chunk fits an IP packet. To distinguish updated content, all chunks of an information item are tagged with a *version identifier* value.

Each user runs an application to request and cache the desired information item. Nodes in the network retrieve the information items from one or more gateway nodes, which permanently cache contents, as well as from other users temporarily storing (part of) the requested items. From time to time, gateway nodes may inject in the network updated versions of the information items. Below, we summarize the basic features of the content retrieval system.

- Each user application requests an information item not in its cache, say information item i ($1 \leq i \leq I$), with rate λ. Note that λ also represents the popularity level of a content. Upon a request generation, the node broadcasts a *query message* for the C chunks of the information item. Then, queries for still missing chunks are periodically issued until the information item is fully retrieved.
- If a node receives a fresh query containing a request for information i's chunks, and it caches a copy of one or more of the requested chunks, it sends them back to the requesting node through *information messages*, one for each owned chunk. Note that a provider node sends back information messages only for cached chunks of the latest content version it is aware of. If the node does not cache (all of) the requested chunks, it can rebroadcast a query for the missing chunks, acting as a forwarder for the request.
- Once created, an information message is sent back to the query source and the only node entitled to cache a new copy of the information is the one which issued the query. In our implementation, information messages are transmitted back to the source of the request in a unicast fashion, along the same path the request came from. In this case, the backtracking follows a node sequence built through information carried by the (forwarded) query messages. Nodes along the way either act as explicit relays for the transiting messages (if they belong to the backtracking node sequence), or simply overhear its transmission without being directly involved in the relaying. Note that, if a node receives multiple information messages corresponding to different versions of the requested content, it will only consider the most recent one and discard the others.
- A node receiving the requested information has the option to cache the received contents and, thus, become a provider for that content to the other nodes. Determining for how long the requested information should be cached is the main objective of this work.

Several optimizations can be introduced to improve the basic scheme for discovery and transfer of contents introduced above. Although our focus is not on query propagation, it is important to take into account the query process as it directly determines the network load associated to the content retrieval operation. We consider two approaches to query propagation:

1) *Mitigated flooding* spatially limits the propagation range of a request by forcing a Time To Live (TTL) for the query messages. Also, it avoids the rebroadcasting of already solved requests by means of a *query lag time*. That is, nodes receiving a query message wait for the so-called query lag time. If during such wait time the nodes observe information messages in reply to the query, they avoid forwarding requests for already obtained chunks.
2) *Eureka* [8] extends mitigated flooding, by targeting queries towards areas of the network where the information is likely to be found. To this end, Eureka lets the users estimate an information density for each information item, in a fully distributed manner. Then, it allows queries to be forwarded towards information-denser areas only, so that only potentially successful requests are propagated.

III. The Hamlet scheme

The process we devise allows users to estimate for how long they should store in their cache an information item they requested and, thus, act as providers for that content. We call this quantity the information *cache drop time*; it is computed separately for each information item and applies to all chunks belonging to that item. Clearly, an information drop time equal to zero means that the user does not store the content in its cache.

The process to estimate the cache drop time is fully distributed and is run by all nodes participating in the network. Its key feature is the fact that it amounts to the node's observation of the *information presence* in its proximity. The procedure does not require any additional signaling exchange among the nodes, but it exploits the observation of query and information messages that are sent on the wireless channel as part of the content sharing application.

We highlight that, although the estimation process requires the nodes to operate in promiscuous mode, only the node that issued the content query is entitled to cache a new copy of that content. A node that overhears an information message caches the content carried in the message only if this is an updated version of the chunks already owned by the node.

In the following, we detail the two main steps of the procedure. First, we describe how a node estimates the presence of information chunks in its proximity, then we outline how the cache drop time (which holds for all chunks belonging to a certain information) is computed.

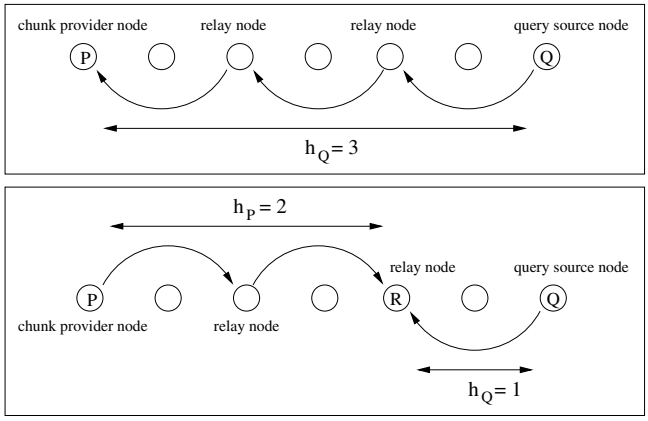


Fig. 1. Q and P denote, respectively, a node issuing a query and a node providing the requested content. Node R in the lower plot is a relay node, overhearing the exchanged messages. The plots represent: case *a)* (upper plot) h_Q value for the provider node P, and case *b)* (lower plot) h_Q and h_P values for relay node R, with respect to the query source Q and the provider P

A. Information presence estimation

Let us define f_e as the frequency at which every node estimates the presence within its *reach range* of each information item it is aware of. We define the reach range of a generic node n as its distance from the farthest node that can receive a query generated by node n itself. The reach range obviously depends on the query TTL and is bounded by the product of TTL and the node radio range.

The generic node n uses the information captured within its reach range, during the estimation step j[1], to compute the following quantities:

- *Provider counter*, $d_{ic}(n,j)$: it accounts for the presence of new copies of information i's chunk c, delivered by n to querying nodes within its reach range, during step j. Node n updates this quantity every time it acts as a provider node (e.g., like node P in the upper plot of Fig. 1);
- *Transit counter,* $r_{ic}(n,j)$: it accounts for the presence of new copies of information i's chunk c, transferred between two nodes within n's reach range and received (or overheard) by n, during step j. Node n thus updates this quantity if it receives (or overhears) an information message (e.g., like node R in the lower plot of Fig. 1) and such message carries chunks belonging to the most recent content version it is aware of.

The provider and transit counters are updated through the hop count information that is included in the query and information message header. The exact procedure is as follows.

a) If node n generates a reply information message containing chunks of information item i, as an answer to a query for some chunk c it owns, then a new copy of such chunks is going to be cached at the node that generated the query. Node n must therefore account for the presence of such new copy at a distance h_Q, which is equal to the number of hops covered by the query (see the upper plot in Fig. 1). The provider counter is updated as follows:

$$d_{ic}(n,j) = d_{ic}(n,j) + \frac{1}{h_Q} \tag{1}$$

Note that the larger the h_Q, i.e., the farthest the new chunk copy, the lesser the added contribution.

b) If node n receives or overhears a new transiting information i message, containing a chunk c whose query status was pending, it must then account for: i) the presence of a new copy of the chunk that will be cached by a node at distance h_Q hops and ii) the presence of an existing copy that is cached at a distance h_P hops (see the lower plot in Fig. 1). Thus, the transit counter is updated as follows:

$$r_{ic}(n,j) = r_{ic}(n,j) + \frac{1}{h_P} + \frac{1}{h_Q} \tag{2}$$

Again, the larger the h_P, i.e., the farthest the information provider, the lesser the contribution of the existing copy.

c) The last case accounts for the reception or overhearing of an information message whose contribution must not (or cannot) be related to a corresponding query. This may happen for two reasons: either the corresponding query was already solved (which means that the message is considered as duplicated information), or no matching query was received (which means that the node moved within transmission range of nodes in the return path after the query was generated and propagated by these nodes). In either case, only the contribution due to the presence of the copy at the provider is considered, hence:

$$r_{ic}(n,j) = r_{ic}(n,j) + \frac{1}{h_P} \tag{3}$$

Based on the above quantities, node n can compute a *presence index* of chunk c of information i, as observed during step j within node n's reach range. We refer to such value as $p_{ic}(n,j)$, and define it as

$$p_{ic}(n,j) = \min\{1, d_{ic}(n,j) + r_{ic}(n,j)\} \tag{4}$$

According to (4), $p_{ic}(n,j)$ is comprised in the range [0,1]. A value zero means that the presence of chunk c of information i was not sensed by n during time step j. Instead, if the chunk is cached one hop away from n, $p_{ic}(n,j)$ is equal to one; this is the "best case" where the chunk would be directly available to n if needed. Intermediate values between 0 and 1 are recorded when n observes chunks being cached more than one hop away. Note that multiple contributions of the last kind can sum up to a maximum information presence $p_{ic}(n,j) = 1$, as we rate a dense presence of chunks a few hops away as valuable as that of a single chunk at one-hop distance.

B. Determining the cache drop time

We denote with $\chi_i(n,j)$ the cache drop time that node n computes at time step j and applies to all chunks belonging to information item i received at time step $(j-1)$.

[1]Note that each estimation step (hereinafter also called time step) has constant duration equal to $1/f_e$

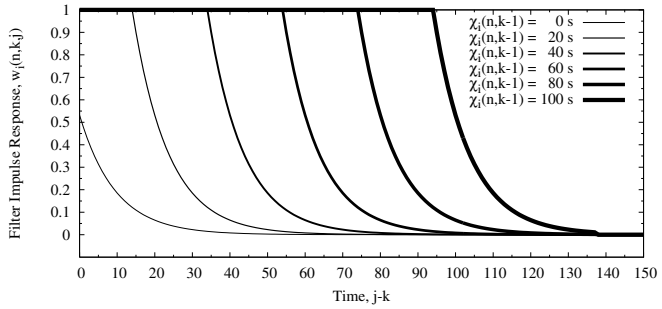


Fig. 2. Filter impulse responses $w_i(n,k,j)$ for different values of $\chi_i(n,k-1)$, and $k=1$, $\alpha=0.9$, $W=0.5$. For $k=1$, the time axis actually marks the time steps since the chunk was cached

To compute $\chi_i(n,j)$, node n composes the presence indices $p_{ic}(n,j)$ of all chunks of information i to an overall probability of information presence, on a per-information item basis, as follows.

Let us first define the amount of time for which the index $p_{ic}(n,j)$ must be considered to be valid. This corresponds to the amount of time for which node n estimates that the copies of the chunk contributing to $p_{ic}(n,j)$ will be cached by the nodes receiving them. If all nodes run Hamlet, the best guess node n can afford to determine the validity of a presence index is to use its local estimate of the cache drop time $\chi_i(n,j-1)$, assuming it is not too different from its neighbors' estimate.

Consistently with this reasoning, the contribution of a presence index computed at step k should only be considered for a time $\chi_i(n,k-1)$. However, discarding contributions *exactly* after a time $\chi_i(n,k-1)$ leads to an *on/off* behavior and yields discontinuities in the caching decision process. Moreover, a hard contribution removal threshold is inconsistent with the uncertainty in the knowledge of the neighbors caching times: the value $\chi_i(n,k-1)$ used by node n may differ from the cache drop time computed by the nodes within n's reach range, especially if they are several hops away.

To account for these factors, we smooth the contributions through an ad-hoc filter. At time step j, node n weighs each past index $p_{ic}(n,k)$, with $k<j$, by a factor $w_i(n,k,j)$, defined as:

$$w_i(n,k,j)=\begin{cases}1 & \text{if } j-k\le\Delta(n,k)\\ \alpha^{j-k-\Delta(n,k)} & \text{else}\end{cases} \quad (5)$$

with

$$\Delta(n,k)=\lfloor f_e\,\chi_i(n,k-1)-\log_\alpha W\rfloor \quad (6)$$

This means that, at time step j, contribution $p_{ic}(n,k)$ maintains its value unchanged if no more than $\Delta(n,k)$ time steps have passed since the presence index was recorded. Otherwise, the filter forces an exponential decrease of the index value. In (5), the exponential decrease factor is denoted by α, while $\Delta(n,k)$ is such that, after a time $\chi_i(n,k-1)$, the smoothed presence index retains a fraction W of its original value[2]. Also, $w_i(n,k,j)$ depends on the information identifier i only, i.e., it has the same value for all chunks c belonging to information i, since the caching time is determined on a per-item basis. For the sake of clarity, examples of filter impulse responses for different values of $\chi_i(n,k-1)$ are shown in Fig. 2.

Next, let us consider time step j and the tagged node n. Node n estimates how complete a single information item i is in its surroundings, due to the contributions measured during a given step k, by summing up smoothed presence indices referring to all chunks c of i:

$$\phi_i(n,k,j)=\min\left\{1, w_i(n,k,j)\frac{1}{C}\sum_{c=0}^{C}p_{ic}(n,k)\right\} \quad (7)$$

In other words, (7) predicts the degree of completeness of information i that node n can expect at time step j, only considering contributions collected during time step k.

The overall presence index for information item i, as seen by node n at time step j, can be computed by summing up the contributions $\phi_i(n,k,j)$ over all steps k:

$$p_i(n,j)=\min\left\{1, \sum_{k=j-\tau}^{j}\phi_i(n,k,j)\right\} \quad (8)$$

Note that in (8) τ is the filter memory, thus contributions older than τ time steps are ignored.

Finally, by denoting the minimum and maximum cache drop time by m_C and M_C, respectively, the caching time for the chunks belonging to information i is obtained as:

$$\chi_i(n,j)=M_C-p_i(n,j)(M_C-m_C) \quad (9)$$

According to (9), in the extreme situation where the entire information i is estimated to be cached within node n's reach range, i.e., $p_i(n,j)=1$, the caching time will be equal to m_C; on the contrary, when node n observes a complete lack of content i within its reach range, i.e., $p_i(n,j)=0$, the caching time will be equal to M_C. Also, if $p_i(n,j)=1$ and $m_C=0$, the retrieved content is not stored by the node.

IV. Simulation Scenario

We tested the performance of Hamlet through the network simulator *ns2*. Hamlet was coupled with mitigated flooding and Eureka, and its performance was compared against that of a deterministic caching scheme, which drops cached chunks after a fixed amount of time, and that of the HybridCache scheme proposed in [2]. We point out that results referring to mitigated flooding (resp. Eureka) coupled with deterministic caching are labeled by "Mitigated" (resp. "Eureka").

In our simulation setup, we consider $I=10$ information items, each comprising $C=30$ chunks. Each query includes 20 bytes plus 1 byte for each chunk request, while information messages include a 20-byte header and carry a 1024-byte information chunk.

A user enters the simulated scenario with an empty cache, and randomly requests any among the I information items not in its cache at a rate given by an i.i.d. Poisson process with parameter λ. The TTL value for query messages is set to 10

[2]Note that multiplying the time interval $\chi_i(n,k-1)$ by f_e allows us to express $\Delta(n,k)$ in number of time steps

hops, while the query lag time is 50 ms. Note that the impact of the query propagation parameters on the information sharing behavior has been studied in [8] and is out of the scope of this paper; here we just consider what previously has been identified as a good parameter setting.

The minimum caching time (m_C) used by Hamlet is set to 0, while the maximum (M_C) is set to 100 s, unless otherwise specified; for the filter parameters, we have $\alpha = 0.9$, $W = 0.5$ and $\tau = 120$. After sampling the parameter space, we have verified that this combination yields the best results, since it provides a smoother behavior of the presence index $p_i(n, j)$. The estimation frequency f_e is set to 1 s^{-1}, however, through extensive simulations, we observed that the impact of f_e is negligible as long as $1/f_e$ is less than 10% of the average caching time.

The information sharing application lies on top of a UDP-like transport protocol, while, at the MAC layer, IEEE 802.11 in promiscuous mode is employed. No routing algorithm is implemented: queries use a MAC-layer broadcast transmission, and information messages find their way back to the requesting node following a unicast path as pointed out in Section II. The channel operates at 11 Mbps, and signal propagation is reproduced by a Two-Ray Ground model.

Finally, we consider two different mobile scenarios: a square city section environment and an L-shaped mall scenario, each featuring two fixed gateway nodes set at opposite ends of the topology (see Fig. 3). Each gateway node permanently stores one half of the information items, the other half being provided by the other gateway. The node radio range is set to 100 m and 25 m in the vehicular and pedestrian scenarios, respectively. We remark that in the city environment this selection of radio range prevents communication between vehicles on parallel, adjacent roads.

In the urban scenario, we simulated a realistic vehicular mobility by means of the Intelligent Driver Model with Intersection Management (IDM-IM), implemented in the Vanet-MobiSim simulator [9], which takes into account car-to-car interactions, stop signs and traffic light presence, and drivers' activity patterns. We simulated a rather sparse traffic with an average number of vehicles equal to 377, travelling over a neighborhood of 6.25 km^2. The mobility model setting led to an average vehicle speed of about 25 km/h, and we observed an average link duration of 24.7 s. The mean and variance of the number of neighbors per node are 6.9 and 5, respectively, and the average number of node clusters is 45, with each cluster being disconnected from the others. In the mall environment, instead, there are on average 128 pedestrian users which move at an average walking speed of 2 km/h, according to the Random Direction mobility model with reflections [10]. The number of neighbors per node has mean and variance equal to 4.4 and 2.5, respectively, and the average link duration is equal to 43 s. The network connectivity level in the mall scenario is significantly higher than in the urban scenario, indeed, on average, there are only 10 disconnected clusters of users.

It is worth pointing out that vehicular mobility in the city environment is characterized by scattered connectivity but high node speed, while the mall environment provides a good network connectivity level, but reduced node mobility. The low mobility of nodes in the mall scenario keeps them away from the sources of unpopular items for long periods of time. Thus, the probability of solving requests for such rare contents is low, unless an efficient caching scheme allows nodes to preserve at least a few copies of every information item in every neighborhood, as Hamlet does.

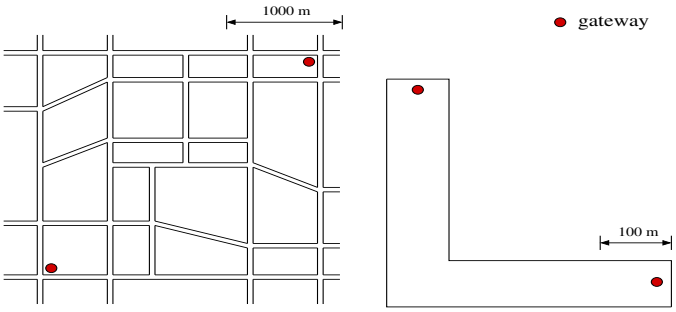


Fig. 3. Simulation scenarios: City (left) and Mall (right)

V. Results

We are interested in two fundamental metrics: the ratio of queries being successfully solved by the system, and the amount of query traffic that is generated. The latter metric, in particular, provides an indication of the system effectiveness in preserving a locally-rich information content: if queries hit upon the sought information in one or two hops, then the query traffic is obviously low. However, it remains to be seen whether such wealth of information is the result of a resource-inefficient, cache-all-you-see strategy, or of a sensible, cooperative strategy such as the one fostered by Hamlet. Therefore, additional metrics, related to cache occupancy and information cache drop time must be coupled with the metrics outlined above.

A. The Hamlet effect

Let us first consider the case where node caches have large enough storage capability to be considered of infinite size and the information contents are not updated, thus only one version per information item lingers in the network. Also, let us set the deterministic caching time to 40 s.

In Fig. 4 we report the ratio between solved and generated information queries (upper plot) and the amount of query traffic (lower plot), in the urban scenario. The results are plotted as functions of the query rate λ, which represents different levels of content popularity.

As can be seen from the plots, both mitigated flooding and Eureka significantly improve their performance as query rates of 0.003 or more are considered. Indeed, when deterministic caching is used, the higher the query rate, the larger the number of nodes caching an information item. This implies that a content can be retrieved with higher probability, and also that it is likely to be found in the proximity of the requesting node,

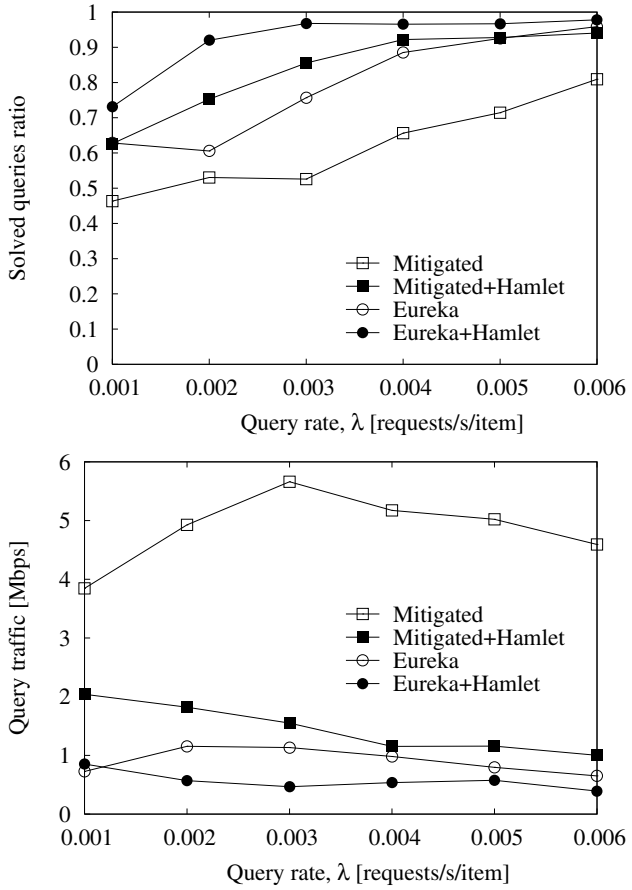


Fig. 4. City scenario: the various schemes are compared in terms of solved queries ratio (upper plot) and query traffic (lower plot), for different values of the content request rate

TABLE I
AVERAGE OCCUPANCY OF THE NODE CACHES, EXPRESSED AS PERCENTAGE OF THE CHUNKS TOTAL NUMBER, FOR $\lambda = 0.003$

Ave. cache occupancy	Mitigated	Mitigated + Hamlet	Eureka	Eureka + Hamlet
City	12.3	8.08	13.8	15.1
Mall	15.6	9.9	14.3	12.2

thus reducing the query traffic per issued request. We also note that, thanks to its efficient query propagation mechanism, Eureka can reduce the propagation of useless queries, hence, collisions among query messages, yielding a higher solved queries ratio than mitigated flooding, for any value of λ.

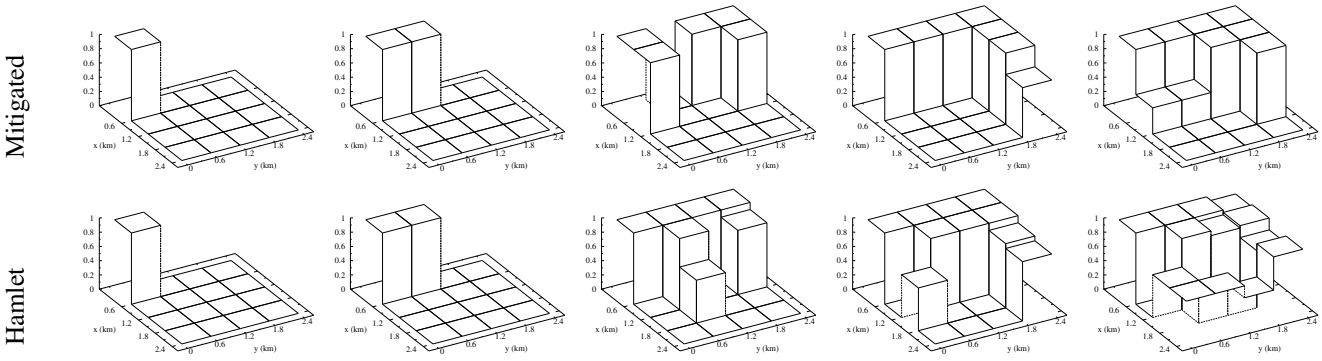
TABLE II
AVERAGE CACHING TIME [S] PER CHUNK WITH $\lambda = 0.003$

Caching Time	Mitigated	Mitigated + Hamlet	Eureka	Eureka + Hamlet
City	40	50.52	40	50.24
Mall	40	45.32	40	55.47

However, from the plots it is evident that deterministic caching strategies used by mitigated flooding and Eureka do not pay off as much as cooperative caching does in Hamlet (whether alone or coupled with Eureka). Hamlet indeed can provide a high solved query ratio, even for λ as low as 0.001. A look at Table I confirms that Hamlet is not even close to a cache-all-you-see strategy, since the average occupancy of node caches is comparable to the values observed in deterministic caching. Results in Table II show that also the caching time per chunk featured by Hamlet is comparable to the one used for deterministic caching. Indeed, it is the quality, not the quantity, of the information cached by Hamlet that allows it to top a sophisticated propagation scheme such as Eureka as far as the solved queries ratio is concerned. The lower plot in Fig. 4 also highlights that the high solved queries ratio provided by Hamlet does not imply an increase in query traffic. On the contrary, Hamlet reduces the overhead both when coupled with mitigated flooding and Eureka by shortening the distance between requesting nodes and desired information content. Further proof of such virtuous behavior by Hamlet is provided in Fig. 5, where the temporal evolution of content presence over the road topology is depicted, for one information item and using mitigated flooding for query propagation. On the one hand, it can be observed that mitigated flooding with deterministic caching creates a sharp separation between the area where the content source resides, characterized by high item availability, and the region where, due to vehicular traffic dynamics, information-carrying nodes rarely venture. On the other hand, Hamlet is shown to favor the diffusion of content over the entire scenario, so that also nodes in areas away from the information source can be served.

Fig. 6 shows the performance obtained in the mall scenario. Looking at the solved queries ratio in the upper plot, we observe that in this case Eureka yields quite poor a performance, while Hamlet can increase the solved queries ratio significantly and, when combined with mitigated flooding, it outperforms all other schemes. The reason why Eureka does not perform well is due to the lack of information items over large areas of the mall scenario, resulting in queries not being forwarded and, thus, remaining unsolved. This is reflected by the curves of query traffic load in the lower plot of Fig. 6. Interestingly, we note that Hamlet greatly reduces the query traffic with respect to mitigated flooding for low as well as high λ's, though providing a much higher solved queries ratio. We also note that, with respect to the urban environment, the mall scenario includes a smaller number of nodes, thus fewer queries are issued and a much smaller amount of query traffic is generated. As for the caching time and occupancy, again, the results in Tables I and II show that in the mall scenario Hamlet leads to results that are comparable with those obtained with deterministic caching, thus proving once more that the performance improvement achieved by Hamlet is due to the more uniform content distribution over the node caches.

Looking at the results in Table II, one may wonder how well Hamlet performs with respect to deterministic caching, when the latter is set to a value other than 40 s. To answer



(a) t = 0s (b) t = 100s (c) t = 200s (d) t = 300s (e) t = 400s

Fig. 5. City scenario: space-time evolution of one information item during the first 400 s of simulation, when mitigated flooding is used with deterministic caching (top) and in combination with Hamlet (bottom). The z axis in the figures shows the content completeness in each spatial slot, a value 1 meaning that the entire content (i.e., all of its chunks) can be found in the space slot

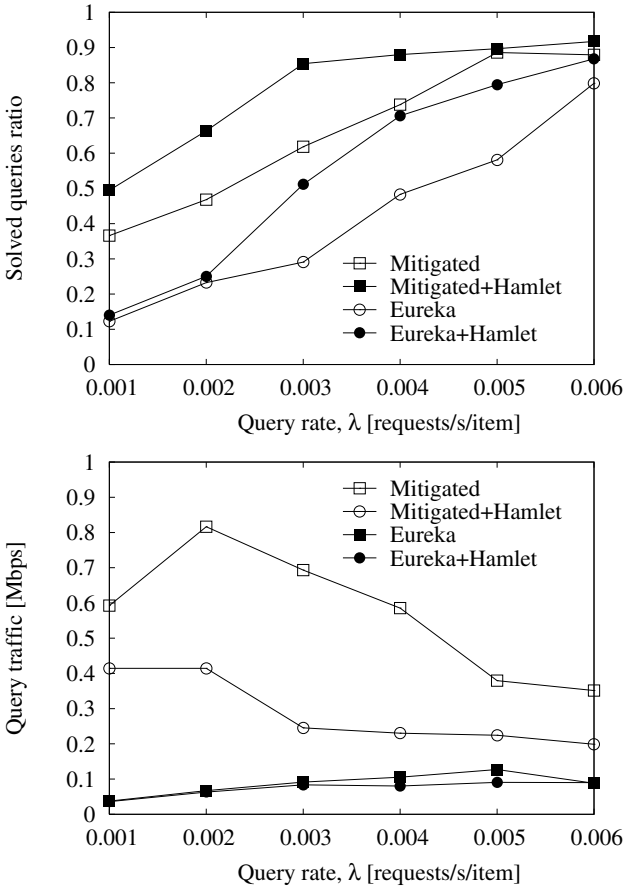


Fig. 6. Mall scenario: the solved queries ratio (upper plot) and the query traffic (lower plot) yielded by the various schemes are presented as functions of the content request rate

this question, we focused on mitigated flooding and, through extensive simulations, we found that deterministic caching achieves the best tradeoff between solved queries ratio and cache occupancy when its caching time is set to the average value of cache drop time obtained using Hamlet. This is a further proof of how Hamlet can autonomously determine optimal operation settings. We therefore derived results using this optimal setting in the urban scenario, i.e., for each λ, the deterministic caching scheme takes as input the average caching time obtained with Hamlet. In particular, Fig. 7 presents the solved queries ratio (upper plot) and the query traffic (lower plot) when Hamlet uses a maximum caching time $M_C = 50$ s and $M_C = 100$ s. Mitigated flooding combined with Hamlet still solves a slightly higher number of queries than mitigated flooding with deterministic caching, and significantly outperforms the deterministic solution in terms of query traffic, due to a better placement of copies. As for the average cache occupancy, for $\lambda = 0.003$ and $M_C = 100$, Hamlet features 8.08% against 18% of occupancy observed under deterministic caching. We can therefore conclude that Hamlet dynamically adapts to the system characteristics and automatically computes an optimal cache drop time.

B. Information consistency

Here, we show some results on the ability of Hamlet to ensure a quick update of the information contents in the network. We assume that one information item is available in the network and that, at some time instants, a gateway node injects an updated version of the item. Also, Hamlet is combined with the mitigated flooding technique for query propagation.

Fig. 8 represents the time diagram of the percentage of chunks of the different content versions cached in the network in the city scenario. Three versions of the information, namely, v_0, v_1 and v_2, are injected at time instants 0 s, 600 s, and 1500 s, respectively. The query rate λ is set to 0.001 (upper plot) and 0.006 (lower plot), which the previous results showed to correspond to, respectively, a very low popularity level

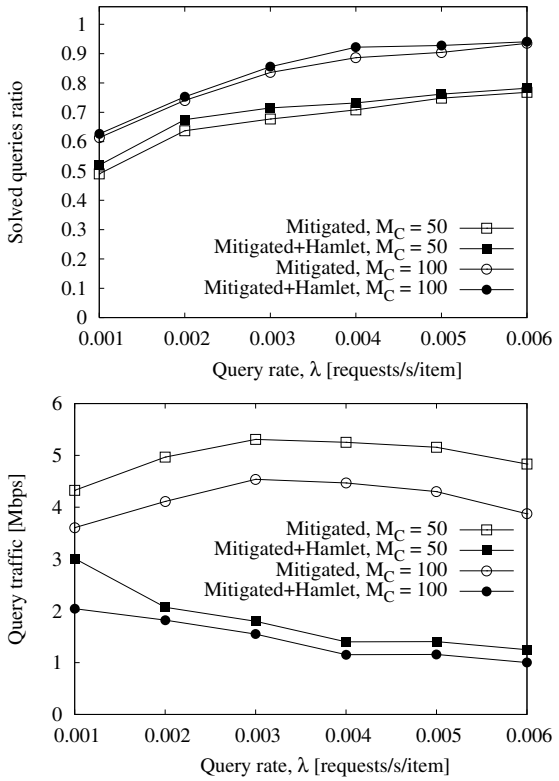


Fig. 7. City scenario: solved queries ratio (upper plot) and query traffic (lower plot) as functions of the query rate, for the mitigated scheme with and without Hamlet. For each λ, the deterministic cache drop time is set to the average cache drop time obtained through Hamlet. Both cases where Hamlet uses $M_C = 50$ s and 100 s are considered

(low solved queries ratio) and a high popularity level (solved queries ratio about equal to 1).

Looking at the results, we note that old versions of the information are rapidly superseded by newer ones, for both low and high popularity levels. Interestingly, the difference in performance as λ varies is negligible; indeed, due to the low connectivity level of the vehicular environment, the system takes about the same time to update all existing copies.

A similar behavior can be observed for the mall scenario, omitted for reasons of space.

C. Comparative evaluation

Finally, we compare the performance of Hamlet with the results obtained by using the well-known HybridCache scheme [2]. According to HybridCache, a node, which requested an information, always caches the received data. Instead, a node on the data path caches the information if its size is small, otherwise it caches the data path, provided that the content copy is not too far away. When the maximum cache size is exceeded, the less popular content is dropped. Note that we assume mitigated flooding to be used for query propagation under both schemes. As a matter of fact, while deriving the results, caching the data path leads to poor performance, due to node mobility; thus we set the HybridCache parameters so that (i) the size of the data never results in data path caching

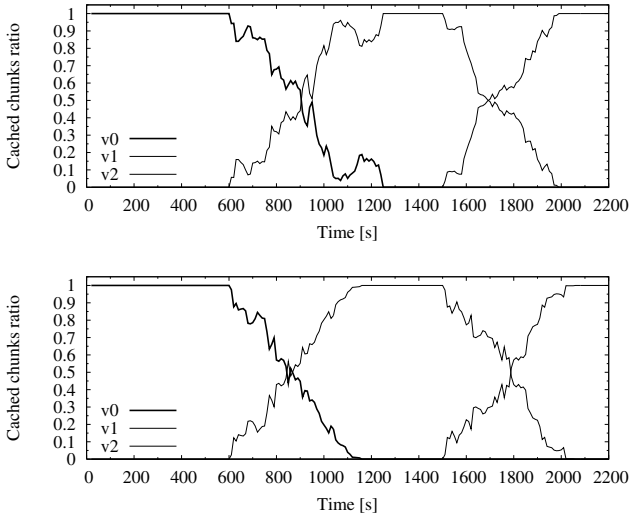


Fig. 8. City scenario: information consistency. Temporal behavior of the percentage of chunk copies of the different information versions in the network, for $\lambda = 0.001$ (upper plot) and $\lambda = 0.006$ (lower plot)

but always in information caching, and (ii) mitigated flooding is employed for query forwarding.

We consider a system in which nodes are provided with a large storage with respect to information size, but cooperative caching is required to use as few buffering resources as possible, since such storage room may be needed by other data, services and applications running at nodes.

Hamlet design allows it to cope with such a scenario without any calibration: nodes autonomously determine their storage needs, by estimating the right amount of buffering (if any) a given information item is worth spending at a certain time, from a cooperative caching perspective.

Instead, the stricter storage policy in HybridCache requires a precise amount of caching that nodes must set aside for cooperative caching. Thus, we performed simulations limiting the cache size of HybridCache, and compared the resulting curves with reference values from a single simulation run with Hamlet, allowing it to self-adapt memory usage to its needs.

Figs. 9 and 10 present the solved queries ratio and the average cache occupancy as the cache size limit imposed to HybridCache varies, in the city and mall scenarios respectively. We express both the cache occupancy and the cache size as a percentage of the amount of information items available to the users, and consider results for two different values of information popularity level (namely, 0.003 and 0.006).

The upper plot in Fig. 9 shows that, in the urban environment, a storage capacity as high as 35% of the available items is needed by HybridCache, if Hamlet performance in query solving is to be matched under any information popularity condition. Such cache size must therefore be set aside by all nodes performing cooperative caching since, as shown by the lower plot of Fig. 9, HybridCache always uses up all the available storage room. Conversely, Hamlet never exceeds a 10% average cache usage, thus freeing more resources for

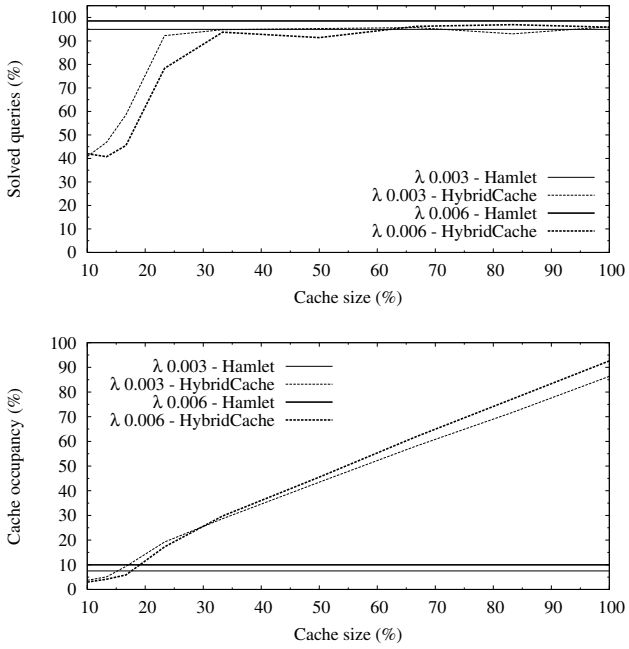


Fig. 9. City scenario: solved queries ratio (upper plot) and average cache occupancy (lower plot) as functions of the node cache size. The performance of Hamlet and HybridCache are compared for $\lambda = 0.003, 0.006$

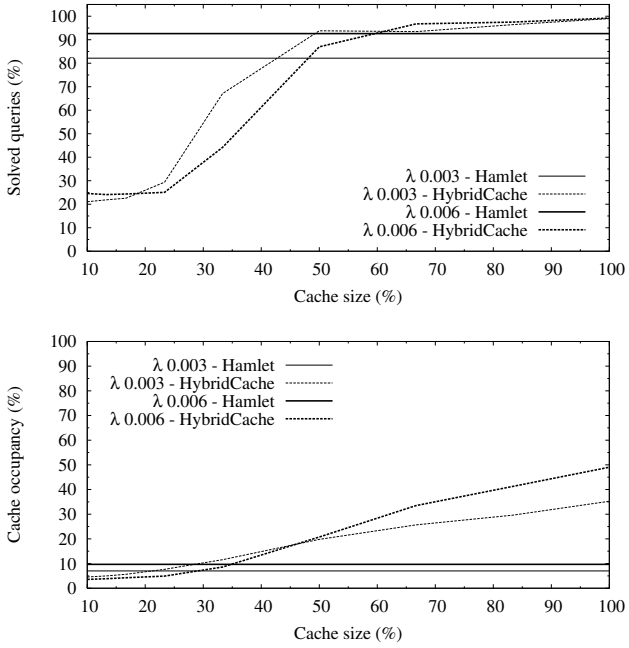


Fig. 10. Mall scenario: solved queries ratio (upper plot) and average cache occupancy (lower plot) as functions of the node cache size. The performance of Hamlet and HybridCache are compared for $\lambda = 0.003, 0.006$

other purposes. In addition, its performance is similar to that obtained by HybridCache even when nodes are allowed to store all the content. Clearly, Hamlet achieves a spatial distribution of information that does not require the residual caching room (representing 90% of the total).

Results for the mall scenario, in Fig. 10, are consistent with those discussed for the urban environment. Indeed, the difference here is even more evident, as HybridCache needs a 60% limit to match Hamlet in solved queries percentage under any content popularity. HybridCache, however, performs better than Hamlet at higher percentage of cache size. From the lower plot in Fig. 10, we also note that the average cache occupancy obtained with HybridCache is lower than in the urban environment; this is because users frequently move out of the mall area and new users (with an empty cache) arrive. Still, the node cache occupancy required by HybridCache is significantly higher than the one required by Hamlet.

VI. Conclusions

We introduced Hamlet, a smart caching strategy for dynamic ad hoc networks whose nodes exchange information items in a peer-to-peer fashion. Hamlet is a fully distributed scheme where each node, upon receiving a requested information, determines the cache drop time of the information depending on the perceived 'presence' of the content in its proximity. The procedure for computing the cache drop time does not cause any additional overhead to the information sharing system. We showed that, thanks to Hamlet's active caching of information that is not held by nearby nodes, the solving probability of information queries is enhanced and the overhead traffic is reduced. Such good performance is obtained for information with low popularity level, as well as for highly popular contents. In addition, Hamlet showed to be able to maintain consistency between copies of the same information item cached at different network nodes.

Acknowledgment

This work was partially supported by Regione Piemonte through the VICSUM project and by the European Union through the Network of Excellence EuroNF.

References

[1] E. J.-L. Lu, and C.-W. Chen, "An Enhanced EDCG Replica Allocation Method in Ad Hoc Networks," IEEE EEE, 2004.

[2] L. Yin and G. Cao, "Supporting Cooperative Caching in Ad Hoc Networks," *IEEE Transactions on Mobile Computing,* vol. 5, no. 1, pp. 77–89, Jan. 2006.

[3] B. Tang, H. Gupta, and S. Das, "Benefit-based Data Caching in Ad Hoc Networks," *Transactions on Mobile Computing,* vol. 7, no. 3, pp. 289–304, Mar. 2008.

[4] T. Hara, "Effective Replica Allocation in Ad Hoc Networks for Improving Data Accessibility," *IEEE INFOCOM,* Anchorage, Alaska, Apr. 2001.

[5] G. Cao, L. Yin, and C. R. Das, "Cooperative Cache-Based Data Access in Ad Hoc Networks," *IEEE Computer,* vol. 37, no. 2, pp. 32-39, Feb. 2004.

[6] T. Hara, "Replica Allocation Methods in Ad Hoc Networks with Data Update," *Mobile Networks and Applications,* vol. 8, no. 4, pp. 343–354, Aug. 2003.

[7] J. Cao, Y. Zhang, G. Cao, and L. Xie, "Data Consistency for Cooperative Caching in Mobile Environments," *IEEE Computer,* pp. 60–66, Apr. 2007.

[8] M. Fiore, C. Casetti, and C.-F. Chiasserini, "Efficient Retrieval of User Contents in MANETs," *IEEE INFOCOM,* Anchorage, AK, May 2007.

[9] M. Fiore, J. Haerri, F. Filali, and C. Bonnet, "Vehicular Mobility Simulation for VANETs," *IEEE Annual Simulation Symposium (ANSS),* Norfolk, USA, Mar. 2007.

[10] E. M. Royer, P. M. Melliar-Smith, and L. E. Moser, "An Analysis of the Optimum Node Density for Ad hoc Mobile Networks," *IEEE International Conference on Communications IEEE ICC*, Helsinki, Finland, June 2001.