Caching Strategies Based on in Sequence Thickness Opinion in Wireless Informal Networks

[1]Chava Kalpana, [2]Mohammad. Shareef
[1,2]Dept. of CSE, Swarana Bharathi Institute of Technology and Science, Khammam, AP, India

Abstract
In the case of small-sized caches, we aim to design a content replacement strategy that allows nodes to successfully store newly received information while maintaining the good performance of the content distribution system.We consider both cases of nodes with large- and small-sized caches. We address cooperative caching in wireless networks, where the nodes may be mobile and exchange information in a peer-to-peer fashion. For large-sized caches, we devise a strategy where nodes, independent of each other, decide whether to cache some content and for how long. Under both conditions, each node takes decisions according to its per- ception of what nearby users may store in their caches and with the aim of differentiating its own cache content from the other nodes'. The result is the creation of content diversity within the nodes neighborhood so that a requesting user likely finds the de- sired information nearby. We simulate our caching algorithms in different ad hoc network scenarios and compare them with other caching schemes, showing that our solution succeeds in creating the desired content diversity, thus leading to a resource-efficient information access.

I. Introduction

Roviding information to users on the move is one of the most promising directions of the infotainment business, which rapidly becomes a market reality, because infotainment modules are deployed on cars and handheld devices. The ubiquity and ease of access of third- and fourth-generation (3G or 4G) networks will encourage users to constantly look for content that matches their interests. However, by exclusively relying on downloading from the infrastructure, novel applications such as mobile multimedia are likely to overload the wireless network (as recently happened to AT&T following the introduction of the iPhone. It is thus conceivable that a peer-to-peer system could come in handy, if used in conjunction with cellular networks, to promote content sharing using ad hoc networking among mobile users. For highly popular content, peer-to-peer distribution can, indeed, remove bottlenecks by pushing the distribution from the core to the edge of the network.

The solution that we propose, called Hamlet, aims at creating content diversity within the node neighborhood so that users likely find a copy of the different information items nearby (regardless of the content popularity level) and avoid flooding the network with query messages. Although a similar concept has been put forward in , the novelty in our proposal resides in the probabilistic estimate, run by each node, of the information presence (i.e., of the cached content) in the node proximity.By lever- aging such a local estimate, nodes autonomously decide what information to keep and for how long, resulting in a distributed scheme that does not require additional control messages. The Hamlet approach applies to the following cases.

A. Large-Sized Caches

In this case, nodes can potentially store a large portion (i.e., up to 50%) of the available information items. Reduced memory usage is a desirable (if not required) condition, because the same memory may be shared by different services and applications that run at nodes. In such a scenario, a caching decision consists of computing for how long a given content should be stored by a node that has previously requested it, with the goal of minimizing the memory usage without affecting the overall information retrieval performance;

B. Small-Sized Caches

In this case, nodes have a dedicated but limited amount of memory where to store a small percentage (i.e., up to 10%) of the data that they retrieve. The caching decision translates into a cache replacement strategy that selects the information items to be dropped among the information items just received and the information items that already fill up the dedicated memory.

The remainder of this paper is organized as follows. First, we discuss the related literature in Section II and outline the system characteristics and assumptions in Section III. The simulation scenarios considered for the performance evaluation of Hamlet are detailed in Section V, whereas the results are presented in Sections VI for large- and small-sized caches, respectively.

II. Related Work

A. Cooperative Caching

In , distributed caching strategies for ad hoc networks are presented according to which nodes may cache highly popular content that passes by or record the data path and use it to redirect future requests. Among the schemes presented in [9], the approach called Hybrid Cache best matches the operation and system assumptions that we consider; Furthermore, the need of a manual calibration of the "cooperation zone" makes the scheme hard to configure, because different environments are considered. Conversely,. The latter approach ensures that the density of different content is proportional to the content's popularity at the system steady state, thus obeying the square-root rule proposed in for wired networks. We point out that the square-root rule does not consider where copies of the data are located but only how many copies are created. It is thus insufficient in network environments whose dynamism makes the positioning of content of fundamental importance and renders steady-state conditions (as assumed in) hard to be achieved. Several papers have addressed content caching and content replacement in wireless networks. In the following sections, we review the works that are most related to this paper, highlighting the differences with respect to the Hamlet framework that we propose.

B. Content Diversity

Similar to Hamlet, in , mobile nodes cache data items other than their neighbors to improve data accessibility. In particular, the solution in aims at caching copies of the same content farther than a given number of hops.. The concept of caching different content within a neighborhood is also exploited in , where nodes with similar interests and mobility patterns are grouped together to improve the cache hit rate, and in , where neighboring mobile nodes implement a cooperative cache replacement strategy. In

both works, the caching management is based on instantaneous feedback from the neighboring nodes, which requires additional messages. The estimation of the content presence that we propose, instead, avoids such communication overhead.

C. Caching With Limited Storage Capability

In the presence of small-sized caches, a cache replacement technique needs to be implemented. Aside from the scheme in, centralized and distributed solutions to the cache placement problem, which aim at minimizing data access costs when network nodes have limited storage capacity, are presented in. which, in mobile networks, need to be maintained similar to routing tables. as well as link-layer traffic monitoring to trigger prefetching and caching. In , the popularity of content is taken into account, along with its update rate, so that items that are more frequently updated are more likely to be discarded. Similarly, in , cache replacement is driven by several factors, including access probability, update frequency, and retrieval delay. These solutions thus jointly address cache replacement and consistency, whereas in this paper, we specifically target the former issue. as will be pointed out, Hamlet can easily be coupled with a dedicated cache consistency scheme.

D. Data Replication

Although addressing a different problem, some approaches to data replication are relevant to the data caching solution that we propose. One technique of eliminating information replicas among neighboring nodes is introduced in [21], which, unlike Hamlet, requires knowledge of the information access frequency and periodic transmission of control messages to coordinate the nodes' caching decisions. In [5], the authors propose a replication scheme that aims at having every node close to a copy of the information and analyze its convergence time.

III. System Outline and Assumptions

Hamlet is a fully distributed caching strategy for wireless ad hoc networks whose nodes exchange information items in a peer-to-peer fashion. cache desired information items.We assume a content distribution system where the following assumptions hold:

- A number I of information items is available to the users, with each item divided into a number C of chunks
- User nodes can overhear queries for content and relative responses within their radio proximity by exploiting the broadcast nature of the wireless medium
- User nodes can estimate their distance in hops from the query source and the responding node due to a hop-count field in the messages.

We detail the features of the specific content retrieval system that we will consider in the remainder of this paper.If a node receives a fresh query that contains a request for information i's chunks and it caches a copy of one or more of the requested chunks, Once created, an information message is sent back to the query source. To avoid a proliferation of information copies along the path, the only node that is entitled to cache a new copy of the information is the node that issued the query. Information messages are transmitted back to the source of the request in a unicast fashion, along the same path from which the request came. Nodes along the way either act as relays for transit messages (if they belong to the backtracking node sequence) or simply overhear their transmission without relaying them. Fig. 1(b) depicts the flowchart of the operations at a node that receives a message that contains an information chunk.

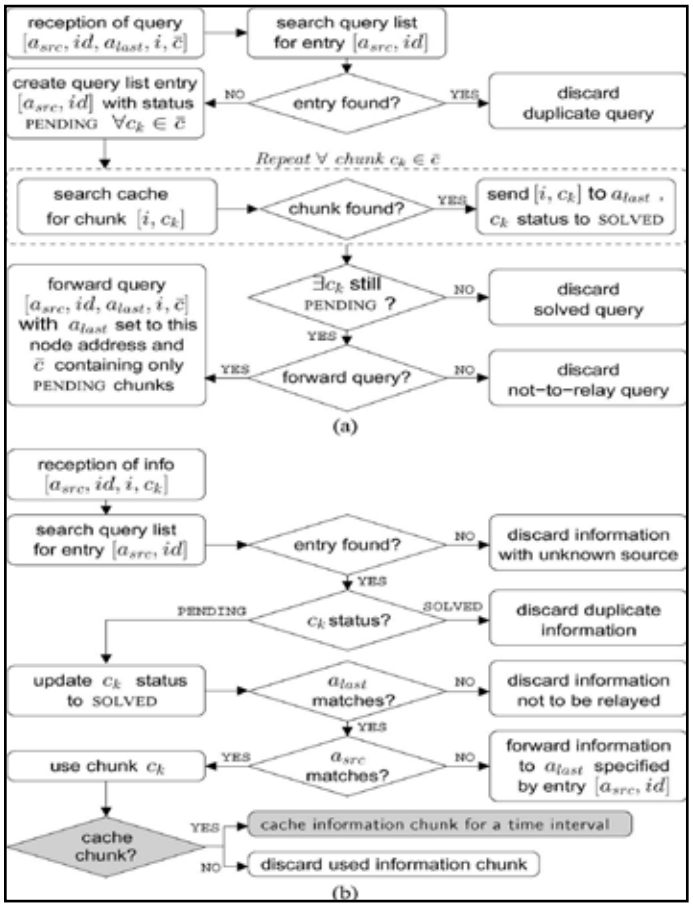

Fig. 1: Flowcharts of the Processing of (a) Query and (b) Information Messages at User Nodes. We Denote the Address of the Node that Generated the Query as asrc , the Query Identifier as id, the Address of the Last Node that Forwarded the Query Message as Alast, and the set of Queried Chunks as $\bar{c}$. The Functional Blocks that are the Focus of this Paper are Highlighted in (b).

A node that receives the requested information has the option to cache the received content and thus become a provider for that content to the other nodes. Determining a strategy of taking such caching decisions is the main objective of this paper, and as such, the corresponding decision blocks are highlighted in Fig. 1(b).

We point out that Hamlet exploits the observation of query and information messages that are sent on the wireless channel as part of the operations of the content-sharing application, e.g., the previously outlined approach. As a consequence, Hamlet does not introduce any signaling overhead.

- Mitigated flooding. This approach limits the propagation range of a request by forcing a time to live (TT L) for the query messages. In addition, it avoids the forwarding of already-solved requests by making the nodes wait for a query lag time before rebroadcasting a query;
- Eureka [23]. This approach extends mitigated flooding by steering queries toward areas of the network where the required information is estimated to be denser.

Note that this paper focuses on cooperative caching and we do not tackle information consistency; thus, we do not take into account different versions of the content in the system model. We note, however, that the previous version of this paper [24] jointly evaluated Hamlet with a basic scheme for weak cache consistency based on an epidemic diffusion of an updated cache content and we showed that weak consistency can be reached, even with such a simple approach, with latencies on the order of minutes for large networks. If prompter solutions are sought,In

the case of Hamlet, a push technique can be implemented through the addition of invalidation messages broadcast by gateway nodes, whereas information providers can pull an updated content (or verify its freshness) before sending the information to querying nodes. In either case, no major modification of the Hamlet caching scheme is required: the only tweaking can consist of resetting the estimation of the information presence upon the notification/ detection of an updated version to ease the diffusion of the new information.

IV. Hamlet Framework

The Hamlet framework allows wireless users to take caching decisions on content that they have retrieved from the network. The process that we devise allows users to take such deci- sions by leveraging a node's local observation, i.e., the node's ability to overhear queries and information messages on the wireless channel. In particular, for each information item, a node records the distance (in hops) of the node that issues the query, i.e., where a copy of the content is likely to be stored, and the distance of the node that provides the information. Based on such observations, the node computes an index of the information presence in its proximity for each of the I items. Then, as the node retrieves content that it requested, it uses the presence index of such an information item to determine whether a copy of the content should be cached, for how long, and possibly which content it should replace. By doing so, a node takes caching decisions that favor high content diversity in its surroundings, inherently easing the retrieval of data in the network.

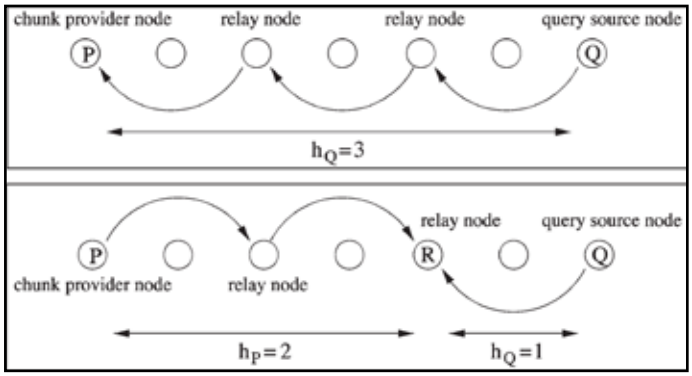

Fig. 2: Q and P Denote, Respectively, a Node that Issues a Query and a Node that Provides the Requested Content. Node R in the Lower Plot is a Relay Node, Overhearing the Exchanged Messages. The Upper and Lower Plots, Respectively, Represent the Case 1 hQ Value for the Provider Node P and the Case 2 hQ and hP Values for the Relay Node R with Respect to the Query Source Q and the Provider P.

whether a copy of the content should be cached, for how long, and possibly which content it should replace. By doing so, a node takes caching decisions that favor high content diversity in its surroundings, inherently easing the retrieval of data in the network.

In the following sections, we first detail how a node estimates the presence of information chunks in its proximity. Next, we separately describe the role of the information presence index in caching decisions for nodes with large- and small-sized caches. In the former case, the information presence index determines the cache content drop time, whereas in the latter case, it drives the cache content replacement.

A. Information Presence Estimation

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. Next, we denote by f the frequency at which every node estimates the presence of each information item within its reach range, and we define as 1/f the duration of each estimation step (also called time step hereafter).

A node n uses the information that was captured within its reach range during time step j to compute the following two quantities:

1. A provider counter by using application-layer data
2. A transit counter by using data that were collected through channel overhearing in a cross-layer fashion. These counters are defined as follows.

- Provider counter dic (n, j). This quantity 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., node P in the upper plot of Fig. 2).
- Transit counter ric (n, j). This quantity 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 infor- mation message,1 e.g., node R in the lower plot of Fig. 2; thus, the transit counter is the only data structure that needs cross-layer access, i.e., the number of information copies whose transit the node has overheard at lower layers (and subsequently inspected).

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 detailed follows:

1. If node n generates a reply information message that contains chunks of information item i, as an answer to a query for some chunk c that it owns, then a new copy of such chunks is possibly cached at the node that generated the query. Node n must therefore account for the presence of such a new copy at a distance hQ , which is equal to the number of hops that were covered by the query (see the upper plot of Fig. 2). The provider counter is updated by a quantity that is inversely proportional to the distance hQ as follows:

Based on the aforementioned 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 a value as pic (n, j) and define it as

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

According to (4), pic (n, j) comprises the range [0, 1]. A zero-value 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, pic (n, j) is equal to one; this case is the "best," where the chunk would directly be available to n if needed. Intermediate values between 0 and 1 are recorded when n observes chunks that are cached more than one hop away. Note that multiple contributions of the last kind can sum up to a maximum information presence $p_{ic}(n, j) = 1$, because we rate the dense presence of chunks a few hops away as valuable as a single chunk at a one-hop distance. The information presence index thus computed plays a crucial role in taking caching decisions in both large- and small-sized caches, as described in the next sections.

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

2. If node n receives or overhears a new transit information

i message, which contains a chunk c whose query status was pending, it must then account for the presence of the following copies: 1) a new copy of the chunk that will be cached by a node at a distance of hQ hops and 2) an existing copy that is cached at a distance of hP hops (see Fig. 2, lower plot). Thus, following the approach in (1), the transit counter is updated as follows:

B. Large-Sized Caches: Computation of the Content Drop Time

We first consider the case in which nodes have a large-sized cache, enough to potentially store a large portion (i.e., 50% or more) of the content that they request. Thus, here, we exploit the aforementioned information presence estimate to determine a cache drop time after which We denote by χi (n, j) the cache drop time that node n
computes at the end of time step j for information item i. Such a
drop time applies to all chunks, belonging to information item i, that will be received during time step (j + 1). To compute χi (n, j), node n estimates an overall probability of information presence, by composing the presence indices pic (n, j) of all chunks of information i, as follows.

Because pic (n, j) are samples of the chunk presence, node n first needs to quantify the amount of time for which these samples are meaningful. If all nodes run Hamlet, the best guess that node n can take to determine the presence index is to use its local estimate of the cache drop time χi (n, j − 1), assuming that it is not very different from its neighbors'. Consistent with this reasoning, the contribution of a presence index computed at step k should only be considered for a time χi (n, k − 1). However, discarding contributions exactly after a time χi (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 χi (n, k − 1) used by node n may differ from the cache drop time computed by the nodes within n's reach range, particularly if they are several hops away.

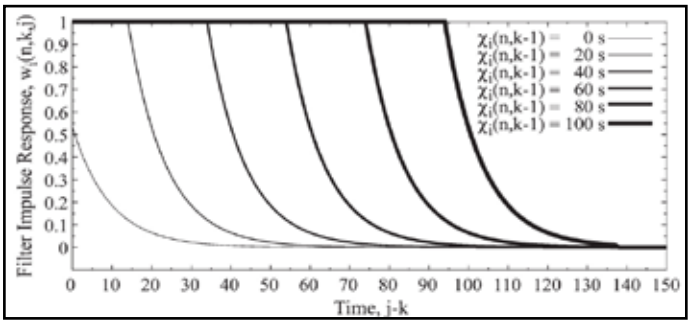

Fig. 3: Filter Impulse Responses w_i (n, k, j) for Different Values of χ_i (n, k −1), where k = 1, α = 0.9, and W = 0.5. For k = 1, the Time Axis Marks the Time Steps Since the Chunk has Been Cached.

To account for these factors, we smooth the contributions through an ad hoc filter. At time step j (whose duration is 1/f), node n weighs each past index pic (n, k), k < j by a smoothing factor wi (n, k, j), which is defined as

C. Small-Sized Caches: Content Replacement

When equipped with a small-sized cache, nodes cannot store all content that they request but are forced to choose which items to keep and which items to discard every time newly retrieved data fill up their memory. In this case, computing cache drop times is clearly not a solution, because the lingering of items in cache is primarily determined by the rate of reception of new content. Therefore, in the presence of limited dedicated storage resources, we exploit the information presence estimate to define a content replacement policy that favors a balanced distribution of data over the network so that all content is as "close" as possible to a requesting node.

The rationale of our content replacement strategy is very similar to the approach employed for the cache drop time computation. Again, we start by identifying the amount of time for which the index pic (n, j) must be considered valid and define a new smoothing factor w^i (n, k, j) to that end as whose details will be discussed at the end of this section, for clarity.

We can thereafter define the completeness of item i, estimated by node n from samples observed at time step k, as

???????????? Eq. is Missing ?????????

and the overall presence index as

$$\hat{p}_i(n, j) = \sum_{k=j-\tau}^{j} \hat{\varphi}_i(n, k, j). \quad (3)$$

With respect to the equivalent formulation for the case of large-sized caches in (7), the index in (11) is not bounded by 1. Indeed, p^i (n, j) is an estimate of the total amount of information i in the reach range of node n at time step j.

We thus leverage p^i (n, j) as the metric for a content replace- ment strategy. Upon the reception, at time step j, of new data to be cached and exceeding the free storage memory, node n discards chunks3 of information i associated with the highest p^i (n, j) until the remaining data fit the storage constraints. In other words, each node tries to keep content that is estimated to be rarer in its surroundings while dropping data that are evaluated to be already commonly available in the area.

How we can compute the estimated caching time χ^i (n, k) remains to be defined. To this end, the best guess that a node can take is to assume that users in its neighborhood attribute similar caching priorities to the information items, i.e., the ordering of p^i (n, k), ∀i, that they compute is in agreement with the ordering of the node. Under this assumption, a node n can estimate the amount of time for which a neighboring user will cache a newly received chunk of item i to be inversely proportional to the p^i (n, k) that it has locally computed. As aforementioned, we define a maximum cache permanence time MC, after which, a chunk is discarded to avoid stored information from becoming stale and node movement from leading to inconsistencies with respect to previous information presence ratings. The estimated caching time is then computed as

$$\hat{\chi}_i(n, k) = \left(1 - \frac{\hat{p}_i(n, k)}{\max_i \{\hat{p}_i(n, k)\}}\right) M_C.$$

To provide an interpretation for (12), consider the case of chunks of the most common information item in the area. Because, as aforementioned, a node discards a chunk of in- formation i associated with the highest p^i (n, j), the estimated caching time for such a chunk is set to 0 in (12), and caching times of much less popular chunks are, instead, estimated to be much longer (up to M_C).

As the estimated presence decreases, the chance that chunks find space in the cache of requesting nodes grows, reaching

the maximum estimated caching time M_c if the information is completely absent from the area, i.e., when $p\hat{}i\ (n, j) = 0$.
Note that (12) does not depend on the content query rate as a precise design choice. Indeed, it is not likely that a node knows the frequency of requests that were generated for each information item by the different users in the network and assuming such knowledge would noticeably limit the feasibility of the solution.

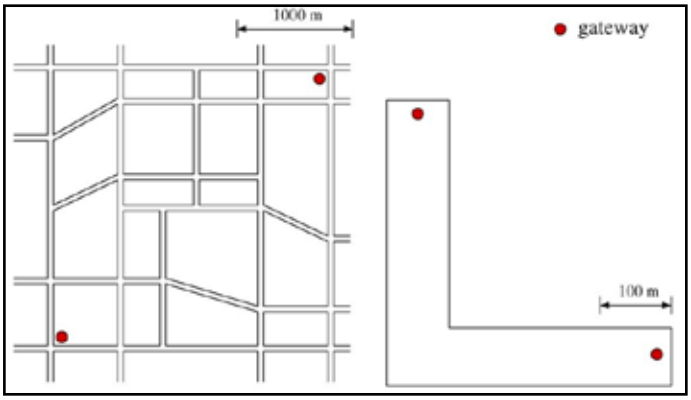

Fig. 4: Simulation Scenarios: City (Left) and Mall (Right)

The performance evaluation that we conducted shows that the lack of query rate awareness does not prevent Hamlet from achieving its goals.

V. Simulation Scenarios and Metrics

We tested the performance of Hamlet through ns2 simula- tions under the following three different wireless scenarios:
1) a network of vehicles that travel in a city section (referred to as City); 2) a network of portable devices carried by customers who walk in a mall (Mall); and 3) a network of densely and randomly deployed nodes with memory limitations (memory- constrained nodes). The three scenarios are characterized by different levels of node mobility and network connectivity.
In the City scenario, as depicted in Fig. 4, vehicle movement is modeled by the intelligent driver model with intersection management (IDM-IM), which takes into account car-to-car interactions and stop signs or traffic lights [27]. We simulated a rather sparse traffic, with an average vehicle density of 15 veh/km over a neighborhood of 6.25 km2 . The mobility model settings, forcing vehicles to stop and queue at inter- sections, led to an average vehicle speed of about 7 m/s (i.e., 25 km/h). We set the radio range to 100 m in the vehicular scenario, and by analyzing the network topology during the simulations, we observed an average link duration of 24.7 s and a mean of 45 disconnected node clusters concurrently present over the road topology. The City scenario is thus characterized by scattered connectivity and high node mobility. The Mall scenario is represented in Fig. 4 as a large L-shaped open space of 400 m of length on the long side, where pedestrian users can freely walk. In this scenario, we record an average of 128 users who walk at an average speed of 0.5 m/s according to the random-direction mobility model with reflections [28]. The node radio range is set to 25 m, leading to an average link duration equal to 43 s, with a mean of ten disconnected clusters of users present at the same time in the network. The connectivity level in the Mall is thus significantly higher than in the City, whereas node mobility is much lower.
The memory-constrained scenario is similar to the sce- nario employed for the performance evaluation of the cache replacement schemes in [9] and [14]. It is composed of 300 wireless nodes deployed over a square area of a side equal to 200 m. Nodes can be static, positioned according to a uniform random distribution, or mobile, wandering according to a random-direction mobility model with reflections. The node speed is uniformly distributed in the range [0.5vm , 1.5vm], where vm is the average node speed—a varying parameter in our simulations. The node radio range is set to 20 m, resulting, for static nodes, in a fully connected network.
In all the scenarios, we deploy two fixed gateway nodes at opposite ends of the topology. Each gateway permanently stores 1/2 of the information items, whereas the other half is provided by the other gateway. Initially, nodes have an empty cache; they randomly request any among the I items that are not in their cache according to a Poisson process with parameter $\lambda i = \Lambda qi \ (1 \leq i \leq I)$. Λ is the query generation rate per node, whereas qi represents the content popularity level (i.e., the probability that, among all possible content, a node requests item i). The TTL value for query messages is set to ten and five hops for the case of large- and small-sized caches, respectively, and the query lag time is 50 ms. Note that the impact of all the aforementioned query propagation parameters on the information-sharing behavior has been studied in [23]; here, we only consider what has been identified as a good parameter setting.
With regard to the Hamlet parameters, the estimation fre- quency is such that $1/f = 0.2M_c$; indeed, through extensive simulations, we observed that the impact of f is negligible, as long as 1/f is not greater than 20% of the maximum caching time. As we fix $\tau = f M_c$, this setting of f leads to a value of τ as small as 5. Then, we have $\alpha = 0.9$ and $W = 0.5$; indeed, we have verified that this combination yields a smoother behavior of the presence index pi (n, j). The values of the remaining parameters are separately specified for large- and small-sized caches.
The information-sharing application lies on top of a User Datagram Protocol (UDP)-like transport protocol, whereas, at the media access control (MAC) layer, the IEEE 802.11 standard in the 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. Information messages exploit the request to send/clear to send (RTS/CTS) mechanism and MAC-level retransmissions, whereas query messages of broadcast nature do not use RTS/CTS and are never retransmitted. The channel operates at 11 Mb/s, and signal propagation is reproduced by a two-ray ground model. Simulations were run for 10 000 s.
In the aforementioned scenarios, our performance evaluation hinges upon the following quite-comprehensive set of metrics that are aimed at highlighting the benefits of using Hamlet in a distributed scenario:

1. The ratio between solved and generated queries, called solved-queries ratio;
2. The communication overhead;
3. The time needed to solve a query;
4. The cache occupancy.

We have further recorded the spatiotemporal distribution of information and the statistics of information survival, because they help in quantifying the effectiveness of Hamlet in preserv- ing access to volatile information. As aforementioned, we did not explore the problem of cache consistency, because such an issue is orthogonal to this paper.

VI. Evaluation with Large - Sized Caches

Here, we evaluate the performance of Hamlet in a network of nodes with large storage capabilities, i.e., with caches that can store up

to 50% of all information items. Because such characteristics are most likely found in vehicular communication devices, tablets, or smartphones, the network environments under study are the City and Mall scenarios. As discussed in Section IV, in this case, the Hamlet framework is employed to compute the caching time for information chunks retrieved by nodes, with the goal of improving the content distribution in the network while keeping the resource consumption low.

We first compare Hamlet's performance to the results obtained with a deterministic caching strategy, called DetCache, which simply drops cached chunks after a fixed amount of time. Then, we demonstrate the effectiveness of Hamlet in the specific task of information survival. In all tests, we assume I = 10 items, each comprising C = 30 chunks. All items have identical popularity, i.e., all items are requested with the same rate $\lambda = \Lambda/I$ by all network nodes. The choice of equal request rates derives from the observation that, in the presence of nodes with a large-sized memory, caching an information item does not imply discarding another information item; thus, the caching dynamics of the different items are independent of each other and only depend on the absolute value of the query rate. It follows that considering a larger set of items would not change the results but only lead to more time-consuming simulations.

Each query includes 20 B plus 1 B for each chunk request, whereas information messages include a 20-B header and carry a 1024-B information chunk. The maximum caching time MC is set to 100 s, unless otherwise specified. Queries for chunks that are still missing are periodically issued every 5 s until either the information is fully retrieved or a timeout that is set to 25 s expires.

A. Benchmarking Hamlet

We set the deterministic caching time in DetCache to 40 s, and we couple DetCache and Hamlet with both the mitigated flooding and Eureka techniques for query propagation. We are interested in the following two fundamental metrics: 1) the ratio of queries that were successfully solved by the system and 2) the amount of query traffic that was generated. The latter metric, in particular, provides an indication of the system effectiveness in preserving locally rich information content: if queries hit upon the sought information in one or two hops, then the query traffic is obviously low. However, whether such a wealth of information is the result of a resource-inefficient cache-all-you-see strategy or a sensible cooperative strategy, e.g., the approach fostered by Hamlet, remains to be seen. Thus, additional metrics that are related to cache occupancy and information cache drop time must be coupled with the aforementioned metrics.

Fig. 5 shows the solved-queries ratio (top plot) and the amount of query traffic (bottom plot) as λ varies in the City scenario. When DetCache is used, the higher the query rate, the larger the number of nodes that cache an information item. This case implies that content can be retrieved with higher probability and also that it is likely to be found in the proximity of the requesting node, thus reducing the query traffic per issued request. Note that, due to its efficient query propagation mech- anism, Eureka reduces the propagation of useless queries (and, hence, collisions), yielding a higher solved-queries ratio than mitigated flooding. However, it is evident that deterministic caching does not pay off as much as cooperative caching does in Hamlet. Table I shows that the average occupancy of node caches in Hamlet is comparable to the values observed with DetCache.

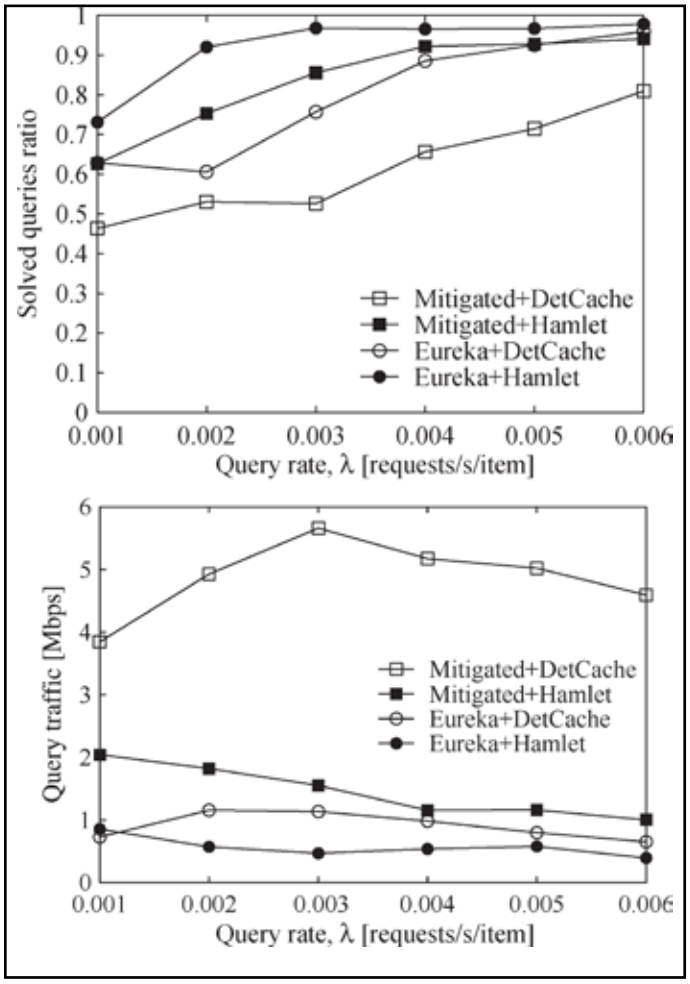

Fig. 5: City: Solved-Queries Ratio (Top) and Query Traffic (Bottom) Obtained with Different Schemes Versus Content Request rate

Table 1: Average Occupancy of the Node Caches, Expressed as A Percentage of the Chunks Total Number for $\lambda = 0.003$

Cache usage	Mitigated + DetCache	Mitigated + Hamlet	Eureka + DetCache	Eureka + Hamlet
City	12.3	8.08	13.8	15.1
Mall	15.6	9.9	14.3	12.2

Thus, 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 positive effect of the caching decisions can also be observed in fig. 5, in terms of the reduced overhead and latency

Table 2: Average Query Solving Time (in Seconds), with $\lambda = 0.003$

Cache Time	Mitigated + DetCache	Mitigated + Hamlet	Eureka + DetCache	Eureka + Hamlet
City	299.442	140.419	157.228	57.617
Mall	24.242	13.795	33.420	24.816

in solving queries. Indeed, Hamlet reduces the overhead by shortening the distance between requesting nodes and desired information content. Similarly, Table II shows how sensible caching choices can significantly reduce the time required to solve queries, again due to the homogeneous availability of information that they generate in the network.

Further proof of such virtuous behavior by Hamlet is provided in

Fig. 6, where mitigated flooding is used for query propagation. The figure depicts the time evolution of content presence over the road topology for one information item; in particular, the z-axis of each plot shows the fraction of different chunks that comprise an information item that are present in a squared area of 600 m2 . On the one hand, it can be observed that mitigated flooding with DetCache 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 favors the diffusion of content over the entire scenario so that nodes in areas away from the information source can also be served.

Fig. 7 refers to the Mall scenario. The poor performance of Eureka in this case 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 [23]. Interestingly, Hamlet greatly reduces the query traffic for any λ, although providing a much higher solved-queries ratio. With regard to the caching occupancy, because Hamlet leads to results that are comparable with the results obtained with DetCache (see Table 1, Mall scenario), it can be asserted that the performance gain achieved through Hamlet is due to the more uniform con- tent distribution across node caches. Finally, Table 2, confirms that such an improved availability of information shortens the waiting time to receive requested items.

When comparing results obtained from the Mall and City scenarios, we note that the solved-queries ratio is consistently lower. We recall that vehicular mobility in the City environ- ment is characterized by scattered connectivity but high node speed, whereas the Mall environment provides a better network connectivity level but reduced node mobility. The low node mobility in the Mall keeps items away from the sources of unpopular items for long periods of time. Thus, the probability of solving requests for such rare content is low, unless an efficient caching scheme allows nodes to preserve at least a few copies of every item in every neighborhood, as Hamlet does. It is also worth pointing out that, with respect to the City environment, the Mall includes a smaller number of nodes; thus, fewer queries are issued, and a much smaller amount of query traffic is generated.

Finally, we may wonder how well Hamlet performs with respect to DetCache when the cache time employed by the latter approach is set to a value other than 40 s. Through extensive

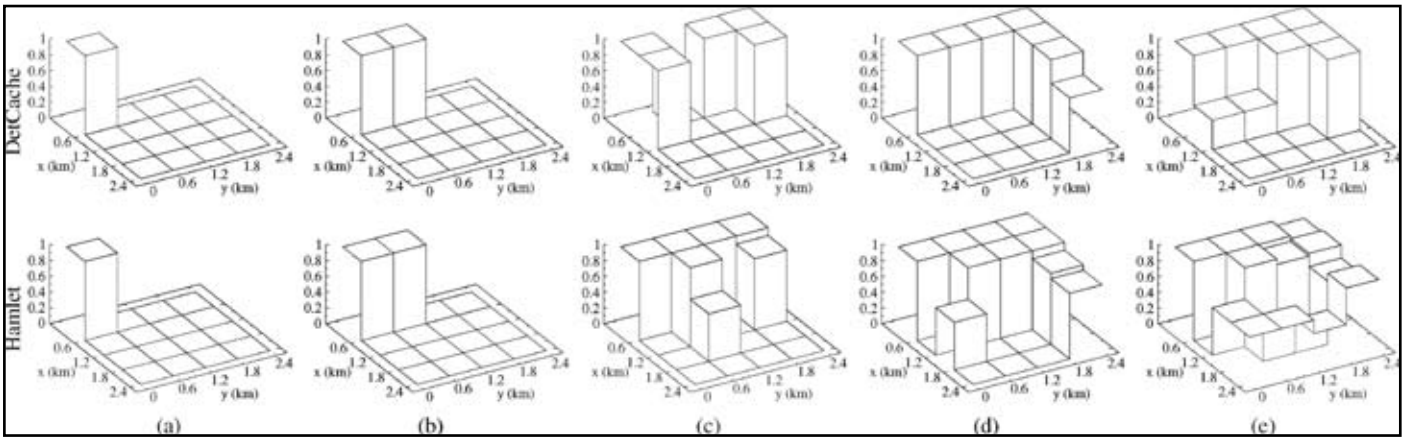

Fig. 6: City: Space–Time Evolution of one Information Item During the First 400 s of Simulation, with Mitigated Flooding and DetCache (Top) and Hamlet (Bottom). The z-Axis shows the Content Completeness in Each Spatial Slot of 600 m^2, with a Value of 1 Meaning that all of the Items' Chunks can be Found in the Slot. (a) t = 0 s. (b) t = 100 s. (c) t = 200 s. (d) t = 300 s. (e) t = 400 s

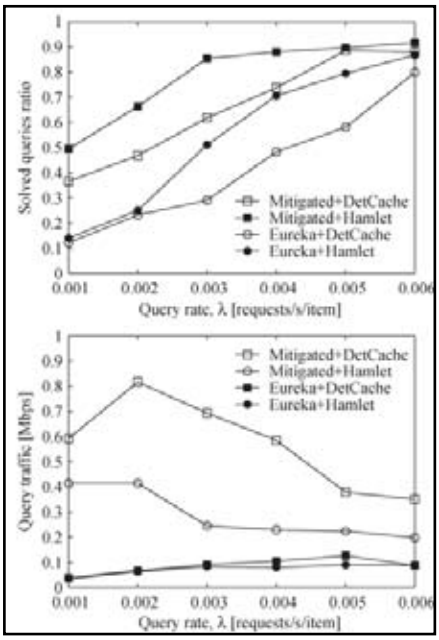

Fig. 7: Mall: Solved-Queries Ratio (Top) and Query Traffic (Bottom) with Different Schemes Versus Content Request Rate

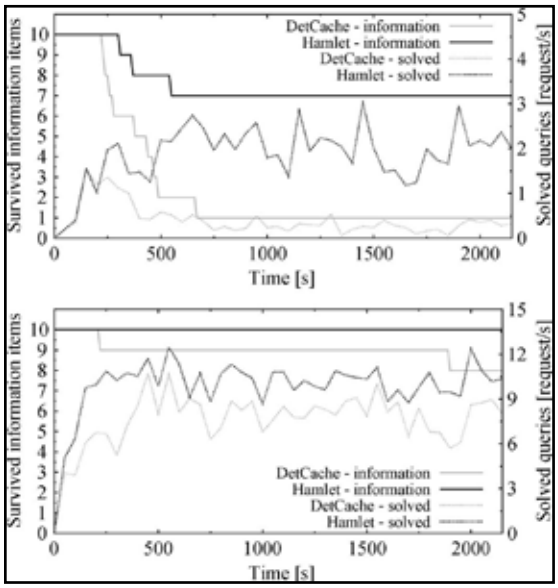

Fig. 8: Information Survival in the Mall (top) and City (bottom) Scenarios. The Temporal Behavior of the Survived Information and Solved Queries when the Gateway Nodes are Switched off at t = 200 s

Fig. 8 shows the time evolution of the number of survived items when the gateway nodes are turned off after t = 200 s in the Mall and the City. The same plots also present the overall solved query rate as time elapses, as observed at 50-s time discretization steps. We consider that DetCache and Hamlet are coupled with mitigated flooding, and we set λ to 0.003 request/s.

In the Mall, the adaptive caching time introduced by Hamlet helps most of the items survive in the system. In the City, the high node mobility supports the circulation of information and, thus, its survival after the gateways shut down. However, even through mobility, DetCache allows some of the items to vanish, whereas Hamlet succeeds in keeping all of items around.

In the Mall, we further explore the reaction of DetCache and Hamlet to different gateway switch-off times. In fig. 9, for each gateway switch-off time, we show the evolution of the number of survived items at some landmark time instants

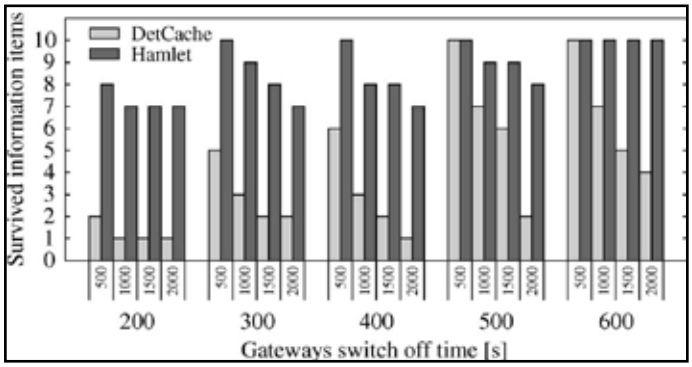

Fig. 9: Mall: Information Survival for Different Gateway Switch-off Times. The Smaller Numbers on the x-Axis Indicate the Landmark Time (in Seconds) to which the Number of Survived Items Refers (computed from the start of the simulation). Clearly, the later the gateways are shut down, the higher the probability of information survival, because the information has more time to spread through the network. We observe that Hamlet can maintain information presence equal to 100% if the information is given enough time to spread, i.e., gateways are disabled after 600 s or more, whereas DetCache loses half the items within the first 2000 s of simulation. We could wonder whether caching times give the edge to either Hamlet or DetCache. However, the average caching time in Hamlet ranges from 37 s to 45 s, depending on the gateway switch-off times and on the specific information item considered. These values are very close to the DetCache caching time of 40 s, showing that Hamlet improves information survival by better distributing content in the network and not by simply caching them for longer periods of time.

VII. EVALUATION WITH SMALL-SIZED CACHES

We now evaluate the performance of Hamlet in a network where a node cache can accommodate only a small portion of the data that can be retrieved in the network. As an example, consider a network of low-cost robots that are equipped with sensor devices, where maps that represent the spatial and tem- poral behavior of different phenomena may be needed by the nodes and have to be cached in the network. We thus consider the memory-constrained scenario introduced in Section V and employ the Hamlet framework to define a cache replacement strategy, as detailed in Section IV.

In such a scenario, the caching dynamics of the different information items become strongly intertwined. Indeed, caching an item often implies discarding different previously stored content, and as a consequence, the availability of one item in the proximity of a node may imply the absence of another item in the same area. Thus, in our evaluation, it is important to con- sider a large number of items, as well as to differentiate among these items in terms of popularity. We consider an overall per- node query rate $\Lambda = 0.1$ and sets of several hundreds of items. We assume that popularity levels q_i are distributed according to the Zipf law, which has been shown to fit popularity curves of content in different kinds of networks [29]. When not stated otherwise, the Zipf distribution exponent is set to 0.5. Such a value was selected, because it is close to the values observed in the real world [29], and the skewness that it introduces in the popularity distribution is already sufficient to make differences emerge between the caching schemes that we study. In any case, we provide an analysis of the impact of the Zipf exponent at the end of this section.

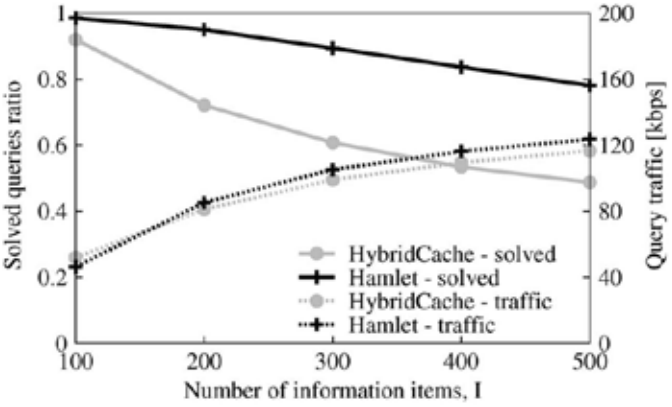

Fig. 10: Static Memory-Constrained Nodes: Solved-Queries Ratio and Query Traffic as the Information Set Size Varies, with Hybrid Cache and Hamlet

We assume that nodes can cache at most ten items, which correspond to a percentage between 2% and 10% of the entire information set, depending on the considered value of I . In ad- dition, we set C = 1 to account for the smaller size of informa- tion items typically exchanged by memory-constrained nodes and MC to 300 s, because the increased network connectivity prolongs the reliability of information presence estimation.

Here, we compare Hamlet with the well-known HybridCache cache replacement technique [9]. In HybridCache, a node that requests an item 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 very far away. When the maximum cache size is capped, content in excess is dropped according to a metric based on the number of requests observed for the different items. Because HybridCache does not exploit information pres- ence estimation, it is less demanding than Hamlet in terms of computation and memory capabilities.

We couple both schemes with mitigated flooding. While deriving the results, we noted that caching the data paths leads to poor performance due to the high cache replacement frequency in the simulated scenarios. Therefore, we set the HybridCache parameters so that the following two conditions are satisfied:

1) The size of the data never results in data path caching but always in information caching, and 2) mitigated flooding is always employed for query forwarding. In addition, to reduce the number of query transmissions in the network, queries for missing chunks are not reissued, and both Hamlet and HybridCache are coupled with the Preferred Group Broadcasting (PGB) technique [30].

A. Benchmarking Hamlet

Let us first focus on the memory-constrained scenario out- lined in Section V with static nodes. Fig. 10 presents the solved- queries ratio and the overall query traffic versus the information set size. We observe that Hamlet reacts better to the growth of the number of items than HybridCache, without incurring any penalty in terms of network load, as shown by the similar query traffic generated by the two schemes.

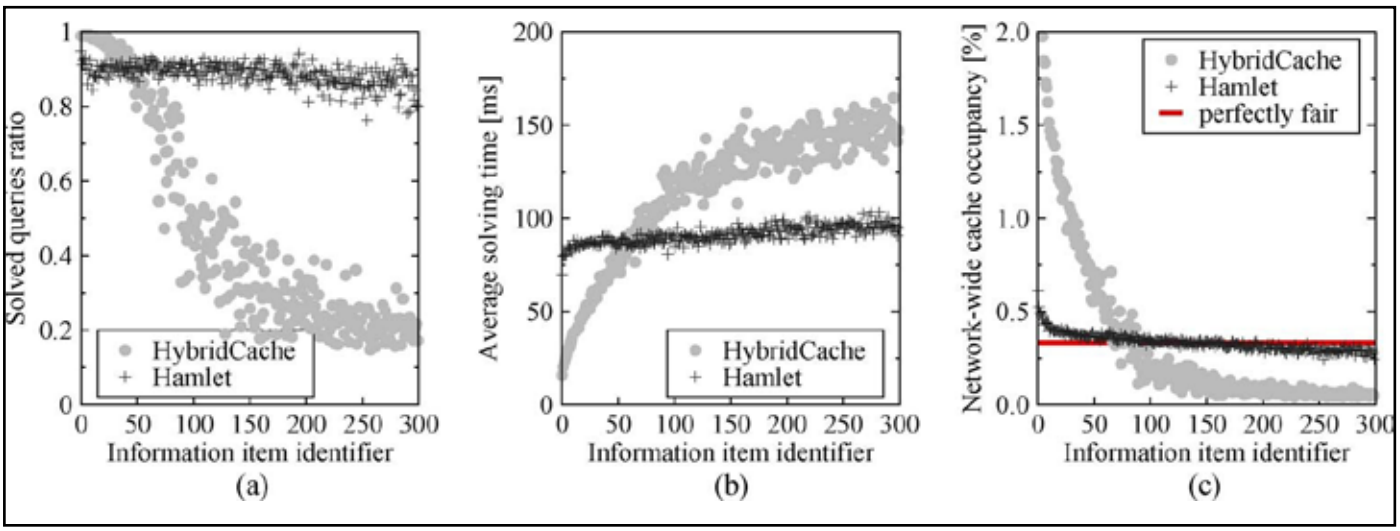

Fig. 11: Static Memory-Constrained Nodes. (a) Query-Solving Ratio, (b) Time, and (c) Average Networkwide Cache Occupancy for Each Item When Using Hybrid Cache and Hamlet, with I = 300. In (c), the Red Horizontal Line Represents Perfect Fairness in cache Occupancy Among Different Items

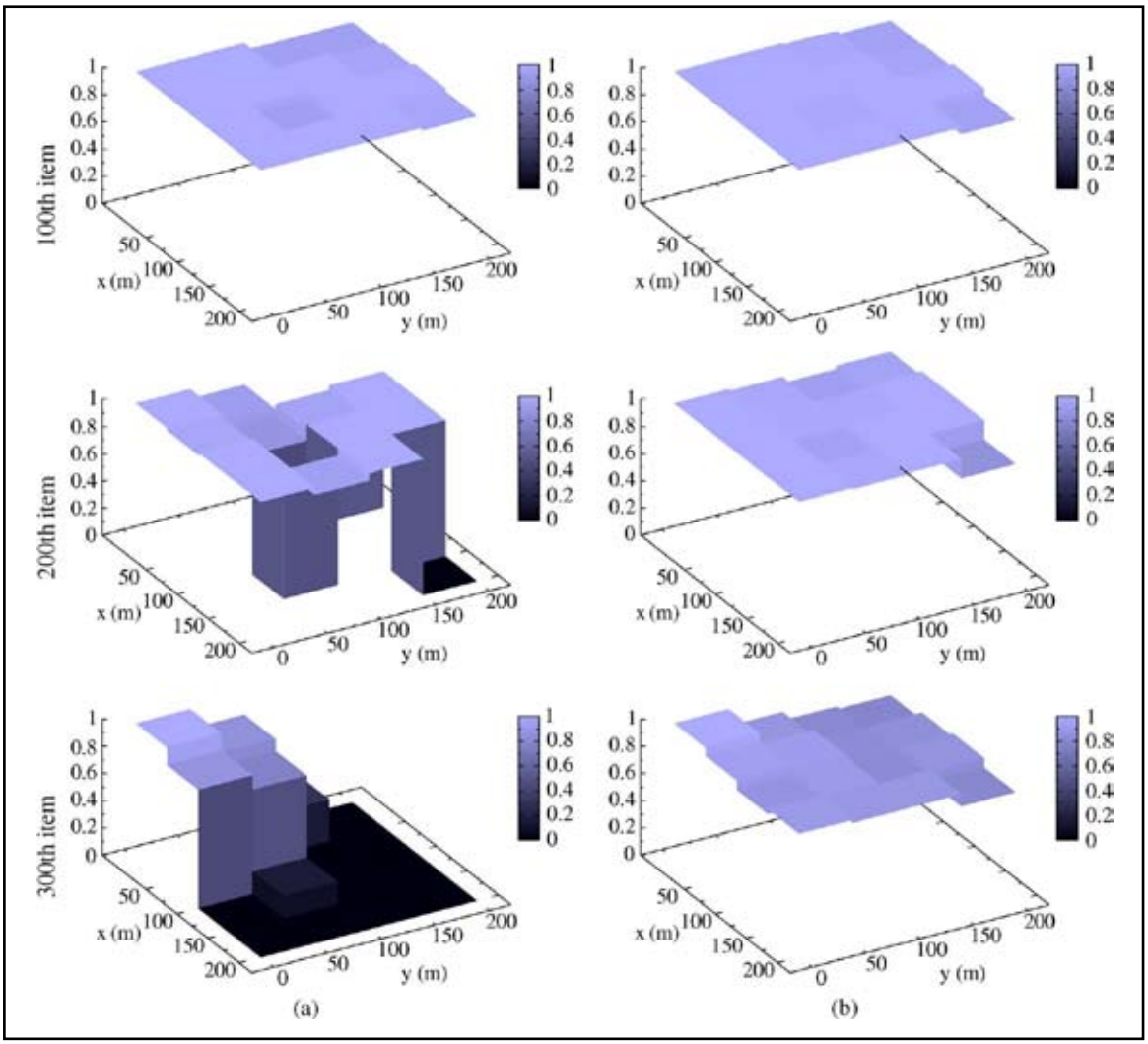

Fig. 12. Static Memory-Constrained Nodes: Spatial Distribution of the 100th, 200th, and 300th Items, Averaged Over Time, for Zipf Distribution Exponents Under HybridCache and Hamlet, with I = 300. The z-Axis in the Plots Shows the Mean Content Completeness in Each Spatial Slot, with a Value of 1, Meaning that the Entire Content can be Found in the Same Spatial Slot. (a) HybridCache. (b) Hamlet

penalty in terms of network load, as shown by the similar query traffic generated by the two schemes.

Observing the performance of Hamlet and HybridCache on a per-item basis allows a deeper understanding of the results. In Fig. 11(a), we show the solving ratio of the queries for each item when I = 300. Along the x-axis, items are ordered in decreasing order of popularity, with item 1 representing the most sought-after information and item 300 the least requested information. Unlike Hamlet, HybridCache yields extremely skewed query solving ratios for the different content; a similar observation also applies to the time needed to solve queries, as shown in Fig. 11(b). The explanation for such behavior lies in the distribution of information in the network. Fig. 11(c) depicts the average percentage of memory used to cache a given item, aggregated over all network nodes. As expected from the previous results, HybridCache fosters the storage of popular content, whereas it disregards content that is less requested, even if it represents two thirds of the whole information set. Instead, Hamlet achieves, in a completely distributed manner, a balanced networkwide utilization of node caches. Indeed, the results of Hamlet are very close to the most even cache occupancy that we can have, represented by the horizontal red line in the plot and corresponding to the case where the total network storage capacity is equally shared among the I items.

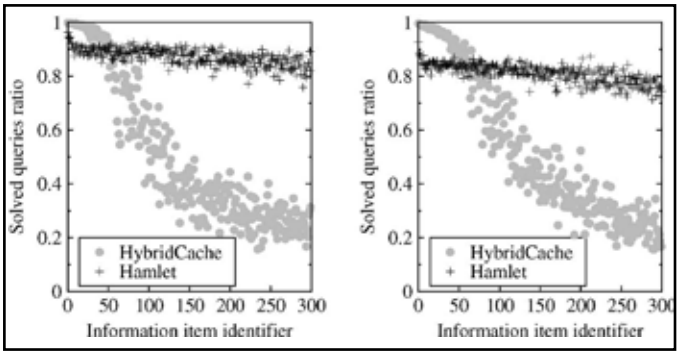

Fig. 13: Memory-Constrained Mobile Nodes: Query-Solving Ratio for Each Information Item when Using HybridCache and Hamlet, with I = 300. The Plots Refer to vm that is Equal to 1 m/s (left) and 15 m/s (right)

Furthermore, it is not only the sheer quantity of data that makes a difference but its spatial distribution also plays a major role. If several nodes cache a rare item but they are all very close to each other, queries that were generated in other areas of the network take more hops to be satisfied. This case happens with HybridCache, as proven by the spatial distribution of the 100th,

200th, and 300th items, as shown in Fig. 12(a). Conversely, the spatial distribution achieved by Hamlet, as shown in Fig. 12(b), is more uniform, leading to a faster more likely resolution of queries.

We now compare the performance of HybridCache and Ham- let in the scenario with memory-constrained mobile nodes. We test the two schemes when I = 300 and for an average node speed vm equal to 1 and 15 m/s.

The solved-queries ratio recorded with HybridCache and Hamlet on a per-item basis are shown in Fig. 13. Comparing the left and right plots, we note that the node mobility, even at high speed, does not seem to significantly affect the results due to the high network connectivity level. The spatial redistribution of content induced by node movements negatively affects the accuracy of Hamlet's estimation process, which explains the slight reduction in the solved query ratio at 15 m/s. That same movement favors HybridCache, at least at low speed, because it allows unpopular information to reach areas that are far from the gateway. However, the difference between the two schemes is evident, with Hamlet solving an average of 20% requests more than HybridCache, when nodes move at 15 m/s.

B. Impact of the Zipf Distribution Skewness

VIII. Conclusion

These decisions are made depending on the perceived "pres- ence" of the content in the node's proximity, whose estimation does not cause any additional overhead to the information sharing system. We have introduced Hamlet, which is a caching strategy for ad hoc networks whose nodes exchange information items in a peer-to-peer fashion. Hamlet is a fully distributed schemewhere each node, upon receiving a requested information, de- termines the cache drop time of the information or which con- tent to replace to make room for the newly arrived information. We showed that, due to Hamlet's caching of information that is not held by nearby nodes, the solving prob- ability of information queries is increased, the overhead traffic is reduced with respect to benchmark caching strategies, and this result is consistent in vehicular, pedestrian, and memory- constrained scenarios. Conceivably, this paper can be extended in the future by addressing content replication and consistency.

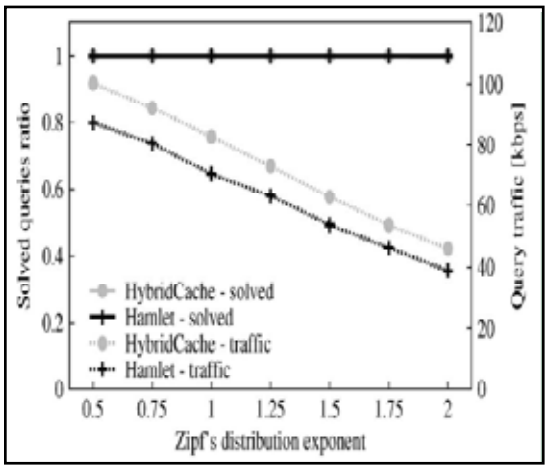

Fig. 14:

The procedure for information presence estimation that was developed in Hamlet can be used to select which content should be replicated and at which node (even if such a node did not request the content in the first place). In addition, Hamlet can be coupled with solutions that can maintain consistency among copies of the same information item cached at different network nodes, as well as with the versions stored at gateway nodes.

References

[1] J. Wortham (2009),"Customers Angered as iPhones Overload AT&T. The New York Times", [Online] Available: http://www.nytimes. com/2009/09/03/technology/companies/03att.html

[2] A. Lindgren, P. Hui,"The quest for a killer app for opportunistic and delay-tolerant networks", in Proc. ACM CHANTS, 2009, pp. 59–66.

[3] P. Padmanabhan, L. Gruenwald, A. Vallur, M. Atiquzzaman,"A survey of data replication techniques for mobile

ad hoc network databases", VLDB J., Vol. 17, No. 5, pp. 1143–1164, Aug. 2008.

[4] A. Derhab, N. Badache,"Data replication protocols for mobile ad hoc networks: A survey and taxonomy", IEEE Commun. Surveys Tuts., Vol. 11, No. 2, pp. 33–51, Second Quarter, 2009.

[5] B.-J. Ko, D. Rubenstein,"Distributed self-stabilizing placement of replicated resources in emerging networks", IEEE/ACM Trans. Netw., Vol. 13, No. 3, pp. 476–487, Jun. 2005.

[6] G. Cao, L. Yin, C. R. Das,"Cooperative cache-based data access in ad hoc networks", Computer, Vol. 37, No. 2, pp. 32–39, Feb. 2004.

[7] C.-Y. Chow, H. V. Leong, A. T. S. Chan,"GroCoca: Group-based peer-to-peer cooperative caching in mobile environment", IEEE J. Sel. Areas Commun., Vol. 25, No. 1, pp. 179–191, Jan. 2007.

[8] T. Hara,"Cooperative caching by mobile clients in push-based information systems", in Proc. CIKM, 2002, pp. 186–193.

[9] L. Yin, G. Cao,"Supporting cooperative caching in ad hoc networks", IEEE Trans. Mobile Comput., Vol. 5, No. 1, pp. 77–89, Jan. 2006.

[10] N. Dimokas, D. Katsaros, Y. Manolopoulos,"Cooperative caching in wireless multimedia sensor networks", ACM Mobile Netw. Appl., Vol. 13, No. 3/4, pp. 337–356, Aug. 2008.

[11] Y. Du, S. K. S. Gupta, G. Varsamopoulos,"Improving on-demand data access efficiency in MANETs with cooperative caching", Ad-Hoc Netw., Vol. 7, No. 3, pp. 579–598, May 2009.

[12] Y. Zhang, J. Zhao, G. Cao,"Roadcast: A popularity-aware content sharing scheme in VANETs", in Proc. IEEE Int. Conf. Distrib. Comput. Syst., Los Alamitos, CA, 2009, pp. 223–230.

[13] E. Cohen, S. Shenker,"Replication strategies in unstructured peer-to- peer networks", in Proc. ACM SIGCOMM, Aug. 2002, pp. 177–190.

[14] B. Tang, H. Gupta, S. Das,"Benefit-based data caching in ad hoc networks", IEEE Trans. Mobile Comput., Vol. 7, No. 3, pp. 289–304, Mar. 2008.

[15] W. Li, E. Chan, D. Chen,"Energy-efficient cache replacement policies for cooperative caching in mobile ad hoc network", in Proc. IEEE WCNC, Kowloon, Hong Kong, Mar. 2007, pp. 3347–3352.

[16] M. K. Denko, J. Tian,"Cross-layer design for cooperative caching in mobile ad hoc networks", in Proc. IEEE CCNC, Las Vegas, NV, Jan. 2008, pp. 375–380.