IOSR Journal of Computer Engineering (IOSRJCE)
ISSN: 2278-0661, ISBN: 2278-8727 Volume 5, Issue 5 (Sep-Oct. 2012), PP 40-46
www.iosrjournals.org

Information Density Estimation in Wireless Ad Hoc Networks Based on Caching Strategies

B.GREESHMA[1], M.Hanock[2], K.SRINIVAS[3]
[1]*P.G.student, KCE, Kurnool Affiliated to JNTUA, Ananthapur.*
[2]P.G.student, KCE, Kurnool Affiliated to JNTUA, Ananthapur.
[3]Professor HOD of Computer Science & Engineering , KCE College, Kurnool.

Abstract: *We address cooperative caching in wireless networks, where the nodes may be mobile and exchange information in a peer-to-peer fashion. We consider both cases of nodes with large- and small-sized caches. For large-sized caches, we devise a strategy where nodes, independent of each other, decide whether to cache some content and for how long. 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. Under both conditions, each node takes decisions according to its perception of what nearby users may store in their caches and with the aim of differentiating its own cache content from the other nodes'. 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.*
Key Words: *Data caching, mobile ad hoc networks.*

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. 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 [2]. 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.

The Hamlet approach applies to the following cases.

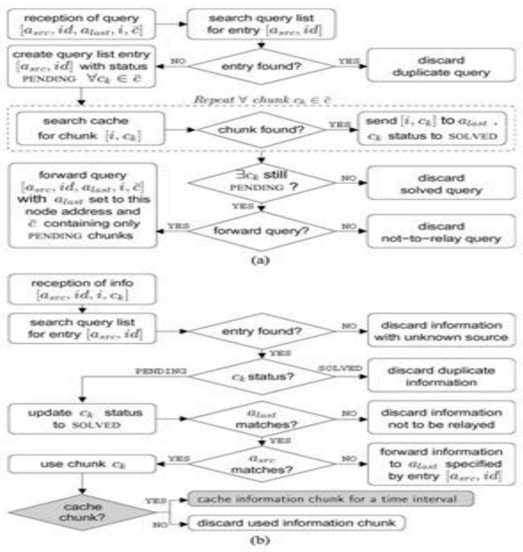
• ***Large-sized caches***. In this case, nodes can potentially
store a large portion (i.e., up to 50%) of the available information items. Reduced memory usage is a desirable (if not required) condition, because the same memory may be shared by different services and applications that run at nodes.

• ***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.

II. 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. We assume a content distribution system where the following assumptions hold: 1) A number I of *information items* is available to the users, with each item divided into a number C of *chunks*; 2) user nodes can overhear queries for content and relative responses within their radio proximity by exploiting the broadcast nature of the wireless medium; and 3) 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. 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, it sends them back to the requesting node through *information messages*. If the node does not cache (all of) the requested chunks, it can rebroadcast a query for the missing chunks, thus acting as a forwarder. The exact algorithm that is followed by a node upon the reception of a query message is detailed in the flow chart in Fig. 1(a).

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. 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).



reception of query $[a_{src}, id, a_{last}, i, \bar{c}]$
search query list for entry $[a_{src}, id]$
create query list entry $[a_{src}, id]$ with status PENDING $\forall c_k \in \bar{c}$
NO
entry found?
YES
discard duplicate query
Repeat $\forall$ chunk $c_k \in \bar{c}$
search cache for chunk $[i, c_k]$
chunk found?
YES
send $[i, c_k]$ to a_{last}, c_k status to SOLVED
$\exists c_k$ still PENDING ?
NO
discard solved query
YES
forward query $[a_{src}, id, a_{last}, i, \bar{c}]$ with a_{last} set to this node address and $\bar{c}$ containing only PENDING chunks
YES
forward query?
NO
discard not-to-relay query
(a)
reception of info $[a_{src}, id, i, c_k]$
search query list for entry $[a_{src}, id]$
entry found?
NO
discard information with unknown source
YES
PENDING
c_k status?
SOLVED
discard duplicate information
update c_k status to SOLVED
a_{last} matches?
NO
discard information not to be relayed
YES
use chunk c_k
YES
a_{src} matches?
NO
forward information to a_{last} specified by entry $[a_{src}, id]$
cache chunk?
YES
cache information chunk for a time interval
NO
discard used information chunk
(b)

III. 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 decisions by leveraging a node's local observation, i.e., the node's ability to overhear queries and information messages on the wireless channel.

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. 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 and 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 $d_{ic}(n, j)$.*** 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 $r_{ic}(n, j)$.*** This quantity accounts for the presence of new copies of information *i*'s chunk *c*, trans ferred 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,[1] e.g., node *R* in the lower plot of Fig. 2

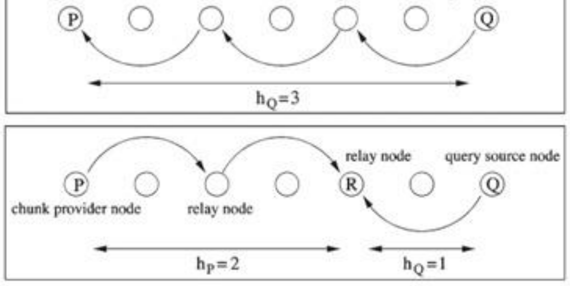


Fig. 2. *Q* and *P* denote, respectively, a node that issues a query and a nodethat provides the requested content.

Node R in the lower plot is a relay node, overhearing the exchanged messages. The upper and lower plots, respectively, represent the case 1 h_Q value for the provider node P and the case 2 h_Q and h_P values for the relay node R with respect to the query source Q and the provider P .

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

The goal is to reduce the cache utilization without affecting the performance of the content distribution system. We denote by $\chi_i(n, j)$ the cache drop time that node n computes at the end of time step j for information item i. To compute $\chi_i(n, j)$, node n estimates an overall probability of information presence, by composing the presence indices $p_{ic}(n, j)$ of all chunks of information i, as follows. Consistent with this reasoning, the contribution of a presence index computed at step k should only be considered for a time $\chi_i(n, k□-1)$. 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, particularly if they are several hops away.
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 $p_{ic}(n, k)$, $k < j$ by a *smoothing factor* $w_i(n, k, j)$, which is defined as

$$w_i(n, k, j) = \begin{cases} 1, & \text{if } j - k \leq \Delta(n, k) \\ a^{j-k\Delta(n,k)}, & \text{otherwise} \end{cases}$$

with $\Delta(n, k) = [fx(n, k\text{-}1) - \log_a W]$

For clarity, examples of filter impulse responses for different values of $\chi_i(n, k□-1)$ are shown in Fig. 3.

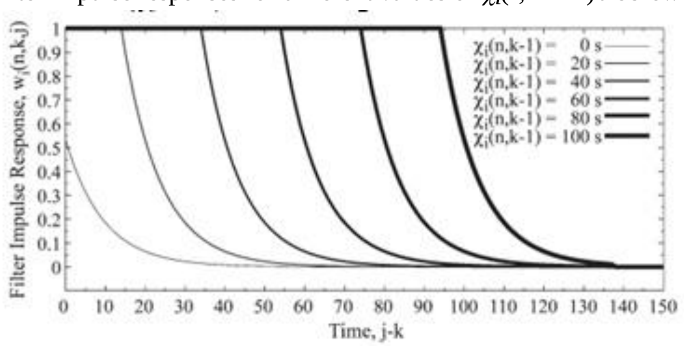


Fig. 3. Filter impulse responses $w_i(n, k, j)$ for different values of $\chi_i(n, k□-□1)$, where $k = 1$, $\alpha = 0.9$, and $W = 0.5$. For $k = 1$, the time axis marks the time steps since the chunk has been cached.

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. we start by identifying the amount of time for which the index $p_{ic}(n, j)$ must be considered valid and define a new smoothing factor $w_i(n, k, j)$ to that end as

$$w_i(n, k, j) = \hat{}\ \begin{cases} 1, & \text{if } j□ - k□ \leq \Gamma(n, k) \\ 0, & \text{otherwise} \end{cases}$$

with $\Gamma(n, k) = f\chi_i(n, k□-1)\ .\hat{}$

the ordering of $\hat{p}_i(n, k)$,□ $\forall i$, that they compute is in agreement with the ordering of the node. The estimated caching time is then computed as

$$\chi_i(n, k) = \hat{}\left(1- \frac{p_i(n, k)}{\max_i \{ p_i(n, k) \}} \right) M_C$$

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 $\hat{}_i(n, j)$, the estimated p 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 chunksfind 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 ˆ$_i(n, j) = 0$

IV. Simulation Scenarios And Metrics

We tested the performance of Hamlet through *ns2* simulations under the following three different wireless scenarios: 1) a network of vehicles that travel in a city section (referred to take 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 memory-constrained scenario is similar to the scenario employed for the performance evaluation of the cache replacement schemes in. 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. 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. 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:

* the ratio between solved and generated queries, called solved-queries ratio;
* the communication overhead;
* the time needed to solve a query;
* the cache occupancy.

V. Evaluation With Large-Sized Caches

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.

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. if queries hit upon the sought information in one or two hops, then the query traffic is obviously low. Thus, additional metrics that are related to cache occupancy and information cache drop time must be coupled with the aforementioned metrics.

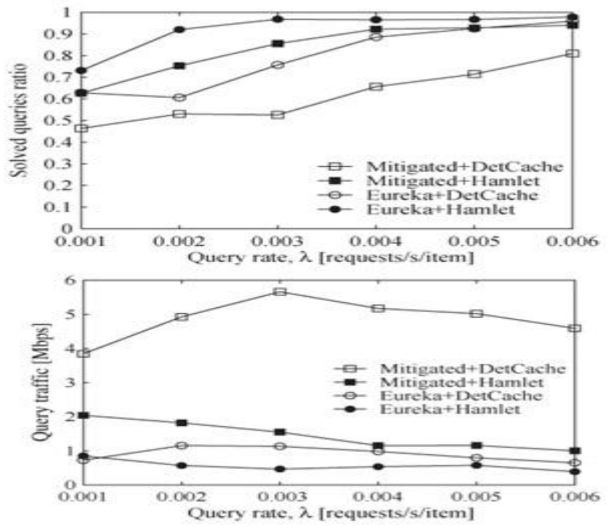


Fig. 4. City: solved-queries ratio (top) and query traffic (bottom) obtained with different schemes versus content request rate. Fig. 5 shows the solved-queries ratio (top plot) and the amount of query traffic (bottom plot) as λ varies in the City scenario. When *DetCache* is used, the higher the query rate, the larger the number of nodes that cache an information item. The positive effect of the caching decisions can also be observed in Fig. 5 in terms of the reduced overhead and latency in solving queries. 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.

Fig. 5 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 With regard to the caching occupancy, because Hamlet leads to results that are comparable with the results obtained with DetCache (see Table I, 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.

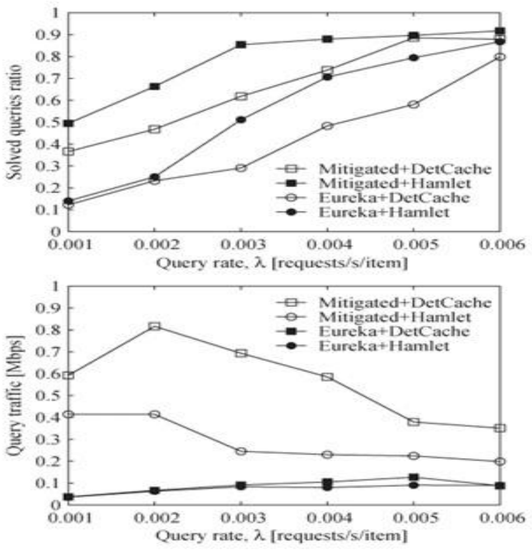


Fig. 6. Mall: Solved-queries ratio (top) and query traffic (bottom) with different schemes versus content request rate.

B. Information Survival

We say that, at a given time instant, a particular item has *survived* if each of its chunks is cached by at least one node in the network. 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.

EVALUATION WITH SMALL-SIZED CACHES

The caching dynamics of the different in- formation 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. 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. 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.

A. Benchmarking Hamlet

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.

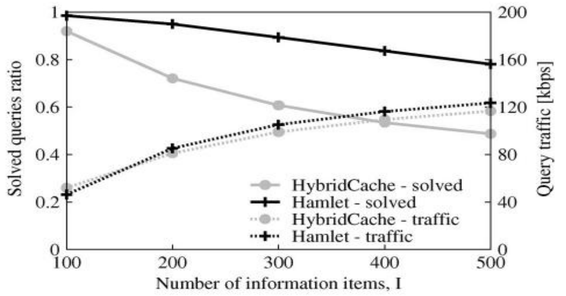
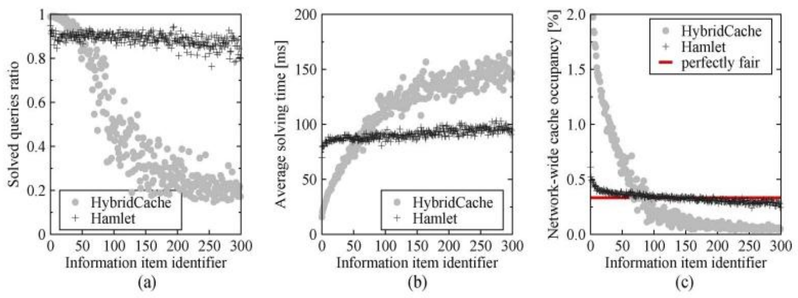


Fig. 7. Static memory-constrained nodes: Solved-queries ratio and query traffic as the information set size varies, with HybridCache and Hamlet.



Observing the performance of Hamlet and HybridCache on a per-item basis allows a deeper understanding of the results. In Fig. 8(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. 8(b). The explanation for such behavior lies in the distribution of information in the network. Fig. 8(c) depicts the average percentage of memory used to cache a given item, aggregated over all network nodes. 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.

This case happens with HybridCache, as proven by the spatial distribution of the 100th, 200th, and 300th items, as shown in Fig. 9(a). Conversely, the spatial distribution achieved by Hamlet, as shown in Fig. 9(b), is more uniform, leading to a faster more likely resolution of queries.

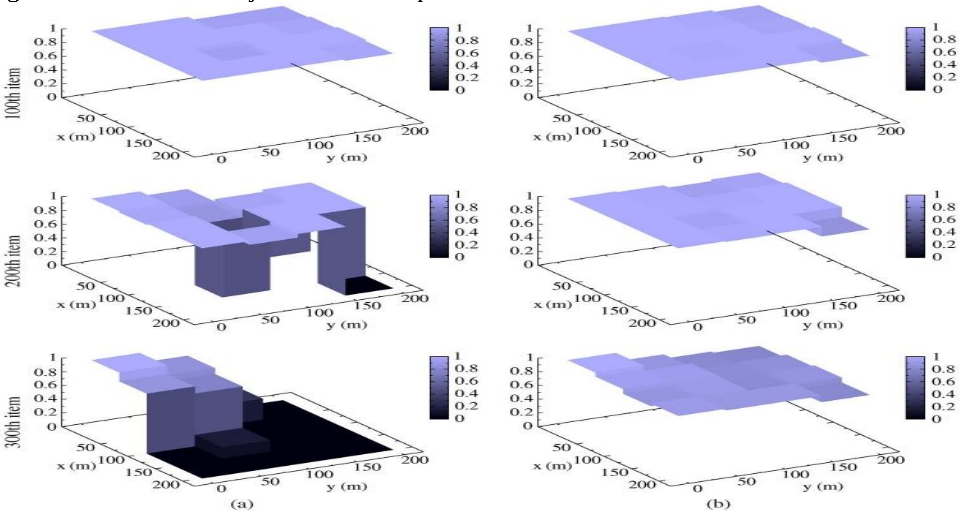


Fig. 9. Static memory-constrained nodes: Spatial distribution of the 100th, 200th, and 300th items, averaged over time, for Zipf distribution exponents under HybridCache and Hamlet, with $I = 300$. The z-axis in the plots shows the mean content completeness in each spatial slot, with a value of 1, meaning that the entire content can be found in the same spatial slot. (a) HybridCache. (b) Hamlet.

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 v_m 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. 10. 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.

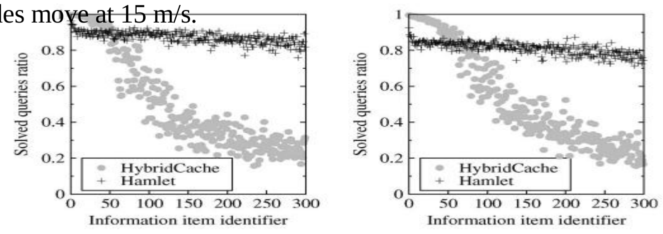


Fig. 10. Memory-constrained mobile nodes: Query-solving ratio for each information item when using HybridCache and Hamlet, with *I* = 300. The plots refer to v_m that is equal to 1 m/s (left) and 15 m/s (right).

B. Impact of the Zipf Distribution Skew ness

We study the impact of the Zipf distribution exponent on the performance of the cache replacement strategies. We recall that an exponent that is equal to zero implies perfect homogeneity, i.e., Zipf distribution that degenerates into a uni form distribution, whereas the difference in popularity among content becomes much more unbalanced as the exponent grows. . The choice of this setting is mandated by the fact that, in the presence of hundreds of different items, unbalanced popularity distributions (i.e., exponents higher than 0.5) lead to very low λ_i for the 100 or so least popular items, thus making requests for such content extremely rare.

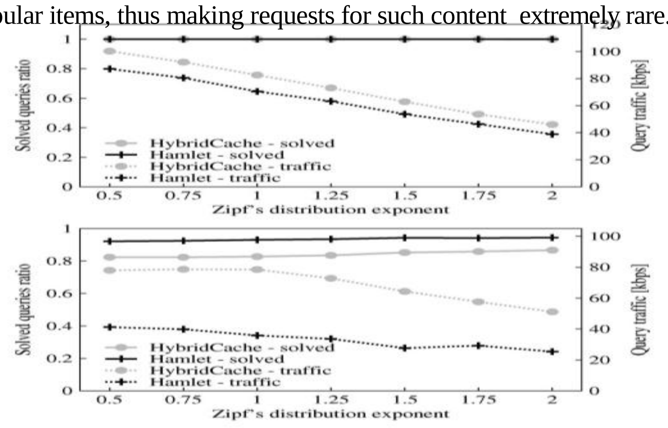


Fig. 11. Memory-constrained static (top) and mobile (bottom) nodes: Solved- queries ratio and query traffic as the Zipf distribution exponent varies when using HybridCache and Hamlet, with *I* = 10.

Fig. 11 depicts the evolution of the solved-queries ratio and the query traffic as the Zipf exponent ranges vary. By comparing the two plots, we note that the presence of mobility (v_m = 1 m/s) leads to a higher number of unsolved requests and in a larger amount of traffic generated within the network under HybridCache, because queries propagate far from the source without finding the desired item. On the one hand, higher values of the exponent lead to more unbalanced query rates, with very few items that are extremely popular and a long tail of seldom-accessed data. On the other, when the Zipf exponent is small, the distribution of queries is more balanced, with information more evenly distributed in the network.

VI. Conclusion

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 scheme where each node, upon receiving a requested information, determines 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 probability 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.

References

[1] A. Derhab and 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.
[2] B.-J. Ko and 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.
[3] C.-Y. Chow, H. V. Leong, and 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.
[4] B. Tang, H. Gupta, and S. Das, "Benefit-based data caching in ad hoc networks," *IEEE Trans. Mobile Comput.*, vol. 7, no. 3, pp. 289-304, Mar. 2008.