

Locating Cache Proxies in MANETs

Roy Friedman*

Maria Gradinariu[†]

Gwendal Simon[‡]

ABSTRACT

Caching Internet based services is a potentially important application for MANETs, as it can improve mobile users' perceived quality of service, reduce their energy consumption, and lower their air-time costs. This paper considers the problem of locating *cache proxies* in MANETs using several search techniques. The paper first examines several existing and a few novel search techniques including flooding, constrained flooding, a novel dynamic variation of probabilistic flooding, and BFS. These are superimposed on a Maximal Independent Set (MIS), a Connected Dominating Set (DS), and a novel adaptation of BFS-tree based overlays, where each of these overlays is maintained in a self stabilizing manner. The paper also includes a comparison of the performance of these search techniques and overlays by extensive simulations.

Categories and Subject Descriptors

H.4 [Information Search and Retrieval]: Miscellaneous

General Terms

algorithms, measurement, performance

Keywords

MANET, Cache Proxy, Distributed Lookup

1. INTRODUCTION

One obvious application domain for MANETs is caching of Internet based (and other network based) services [9]. Caching is a common technique used in various areas of computer science to

*Computer Science Departement, Technion, Haifa 32000, Israel, e-mail: roy@cs.technion.ac.il. Most of the work was done while the author was visiting IRISA. This author is partially funded by an IBM Faculty Partnership Award

[†]Irissa, Universite Rennes1, France, email: mgradina@irisa.fr

[‡]France Telecom R&D, Universite Rennes 1, France. email: gwendal.simon@rd.francetelecom.com.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

MobiHoc'04, May 24–26, 2004, Roppongi, Japan.

Copyright 2004 ACM 1-58113-849-0/04/0005 ...\$5.00.

improve performance, scalability, and quality of service when some form of locality exists. When considering network based services offered to mobile clients, it is likely that multiple clients in the same MANET, or even the same region of a MANET, will try to access the same service concurrently. This suggests that caching such services within the MANET would be beneficial.

As in traditional Web caching, we refer to the node that caches the service as a *local proxy*. The two main issues in any such caching system are *proxy placement* and *proxy lookup*. The former deals with deciding which nodes should act as proxies for which services and the latter solves the issue of how to find a proxy and how to route to it.

Obviously, choosing the right proxy can have a dramatic impact on the performance of the system [9]. For example, the number of proxies for each service in a MANET should be balanced in order to minimize external communication, minimize internal communication, and maximize the number of services that can be cached within the network in a given time. These need to be updated continuously to make up for topological changes and partitioning and merges of the MANET caused by mobility, but with minimal overhead. When cellular communication is used to access the Internet, the connection quality and location of each node's service provider within the Internet should be taken into account as well.

Yet, having an efficient and effective lookup scheme is just as important for the success of a caching mechanism, as without it, the proper proxies will not be relevant. Thus, this paper deals with cache proxy lookup. In particular, the solutions we strive for in this work must be completely decentralized due to the ad-hoc nature of the environment, must be able to cope with continuous changes in the underlying network topology and proxy placements, and be communication efficient. Thus, we focus on search based lookups rather than on maintaining directories. Specifically, we explore several search dissemination techniques, including unconstrained flooding, constrained flooding along the arcs of a logical overlay topology, probabilistic flooding, and breadth first search (BFS). Moreover, for probabilistic flooding, we offer a novel mechanism for adapting the forwarding probability dynamically based on the observed system behavior, which extends the work of [17, 25]. These four methods were chosen as they represent most of the common distributed search techniques.

Additionally, we explore various approaches to constructing the overlay on which the dissemination occurs using both proactive and reactive methods. These include a *Maximal Independent Set* (MIS) based construction,¹ a *Connected Dominating Set* (DS) based construction, and a novel adaptation to MANETs of a *BFS-tree* based construction initially published in [33] (in the context of fixed net-

¹This is also known as *weakly connected maximal independent set* (WCNIS) in [2].

works). The idea in all these schemes is to construct an overlay that contains a small number of nodes, but still obtain full coverage of the network while only relying on local exchanges of information and knowledge. The proposed algorithms are fully decentralized and are designed in a self-stabilizing manner in order to cope with transient faults and mobility. In particular, they do not need to be (re)initialized after a fault or a physical topology change. Whatever the initial configuration is, the algorithms satisfy their specification after a stabilization period [7]. Our construction of the MIS based overlay is novel, and consists of two parallel tasks, namely, computing the MIS nodes and adding bridge nodes between them. Also, a nice feature of the BFS-tree scheme is its *snap stabilization*, i.e., the stabilization time of the proposed scheme is zero.

For motivating these schemes, it was shown in [15] that MIS has the appealing property in MANETs that any node not in the MIS has at most 5 neighbors in the MIS. This means that the number of redundant messages forwarded by an MIS based overlay is small. Similarly, in a dominating set based overlay the number of redundant messages is also expected to be small, but the construction requires exchanging much larger messages, as it needs information on all neighbors at distance 2 hops. Finally, the BFS-tree based scheme is a reactive scheme we introduce in this work that exploits the BFS search properties in order to construct an efficient dissemination tree (it can be viewed as an adaptation of the work of [33] to MANETs). It is therefore expected to result in very little redundant messages. On the other hand, it is expected to impose higher search latencies and is limited to a BFS search style.

Next, we compare several combinations of the dissemination techniques with the overlay maintenance schemes by extensive simulations. In these simulations we measure the *hit-ratio* obtained by each method, the *average latency* for obtaining a reply, and the number of messages generated due to each request. This is done for varying system parameters such as transmission range, nodes' velocities, hot-spot probability, and proxy density. The hit-ratio is an indication of the probability that a method will find a proxy if one exists, or in other words, for its effectiveness. The average latency is clearly of interest to users, as one of the main motivations for caching is improving access latencies to data. Finally, the number of messages generated by each search request is the cost imposed by each method, both in terms of resources (energy, bandwidth, etc.) and in terms of potential scalability.

The results of our simulations, as detailed in the paper and summarized in Section 6, shed light on the behavior of these methods. In particular, they indicate some general patterns common to BFS-like searches vs. flooding based searches, and proactive based overlays vs. truly reactive schemes. They also provide some insight for developing cache placement techniques, as they imply the existence of an optimal number of proxy replicas (for the same data item). This number ensures that one of these proxies will be found with very high probability, but adding more proxy replicas has little added benefit. This replication level is also optimal for time latency and for the communication efficiency of BFS-based schemes.

1.1 Related Work

The problem of cache lookup in ad-hoc networks is fundamentally different than in wired networks. Nodes mobility leads to frequent disconnections of wireless links, hence the network topology changes dynamically. Moreover, the absence of a hierarchical organization of the network renders useless many classical techniques proposed for the Internet, e.g., [1, 3, 4, 8, 16, 26, 30] (to name a few).

Largely speaking, there are three types of routing strategies for MANETs, namely, *proactive protocols*, *reactive protocols*, and *hy-*

brid protocols. Proactive protocols, e.g., OLSR [5], periodically update their routing tables, and thus always maintain (possibly implicit) routes from any node to any node. Reactive protocols, e.g., AODV [20] and DSR [13], discover a route only when it is needed. This way, they do not waste resources on routes that are not needed and would never be used, but their response time is much slower and their route discovery process is typically communication inefficient. Hybrid protocols, e.g., ZRP [31], have a proactive behavior in the neighborhood of a node up to a distance k . After that distance, the protocol acts as a classical reactive protocols. Thus, they attempt to enjoy both worlds. Inspired by the above classification, in this work we propose proactive and reactive lookup algorithms. The proactive lookup assumes that the neighborhood of a node is known in advance, e.g., due to a proactive *heartbeat* mechanism, while the reactive class constructs the neighborhood at every reincarnation of a query.

Broadcast is one of the most popular services on top of MANETs and is highly related to cache lookup. The main difference between broadcast and cache lookup is their goal: broadcast tries to ensure delivery of messages to all processors in the ad-hoc network, while lookup should provide a path to the service or proxy if one exists in the network. In [27] the authors analyze several different broadcast techniques, including: simple flooding [12], probability based methods [17], location (and area) based schemes [17], a scalable broadcast algorithm [18], and an ad-hoc broadcast protocol [19]. The authors of [27] analyze the following simulation parameters: delivery ratio, number of retransmitting nodes, end-to-end delay, and total packets transmitted per node per broadcast. In the current work we take a similar approach by analyzing the lookup algorithms for the number of messages they generate per request, their hit ratio and their lookup time latency.

Topology discovery is another service related to cache lookup. The main issue here is to collect all topology related information and to distribute it to all the nodes in the network. The techniques applied in [22] use mobile agents for information dissemination and the notions of link stability and aging in order to predict the current network topology.

The work of [9] motivates the necessity of cache placements in ad-hoc networks. It also presents many ideas for efficient implementation of placement and lookup services for such a caching mechanism.

Recent developments of peer-to-peer schemes may advocate for their use in the context of ad-hoc networks. In particular, existing peer-to-peer schemes like Pastry [21], Tapestry [32], and Chord [24] are decentralized and communication efficient. Yet, these schemes do not react well to frequent changes in the network topology and/or proxy placements. Nevertheless, some ideas like, for example, assembling nodes based on their similarity, may be applied in the context of ad-hoc networks. Hence, in [23] the authors propose for the first time an energy based study of caching strategies for mobile terminals interacting via an ad-hoc network and using a peer-to-peer approach. The lookup is based on a ZRP-like algorithm where the nodes within a zone are contacted using a broadcast strategy and the nodes outside the zone are contacted using a peer-to-peer strategy. In order to prune the search process, the authors use a similarity function. The experimental evaluation of the proposed solutions is conducted using energy as the cost measure.

2. SYSTEM MODEL AND DEFINITIONS

In this work we focus on wireless mobile systems. A *node* in the system is a device owning an omni-directional antenna that enables wireless communication. A transmission of a node p is received by all nodes within a disk centered on p whose radius depends on

the transmission power, referred to in the following as the *transmission disk*; the radius of the transmission disk is called the *transmission range*. Thus, there is a single communication primitive, $\text{broadcast}(m)$, allowing a node to transmit a message m to all nodes inside its transmission disk. The combination of the nodes and the transitive closure of their transmission disks forms a wireless ad-hoc network.

Practically, in order to obtain external services, we assume that all nodes have another mean for accessing the Internet directly, e.g., using a cellular connection or a Wireless Access Point (WAP); such an access to the Internet passes through an Internet Service Provider (ISP). However, the details of these are mainly of concern for proxy placement, and thus we ignore them in the rest of this work. We say that a node p is a *local proxy* for a data object d when the buffer cache of p contains d .

The network described above can also be modeled as a graph $G = (V, E)$ where V is the set of network nodes and E models the one-to-one neighboring relations. A node q is a *neighbor* of another node p if q is located within the transmission disk of p . In the following, $\mathcal{N}(p)$ refers to the set of neighbors of a node p . By considering $\mathcal{N}(p)$ as a relation (defining the set $\mathcal{N}(p)$), we say that a node p has a *path* to a node q if q appears in the transitive closure of the $\mathcal{N}(p)$ relation.

As nodes can physically move, there is no guarantee that a neighbor at time t will remain in the transmission disk at a later time $t + \Delta$. Messages are not guaranteed to be delivered, but we assume the most of them are delivered with high probability. Additionally, devices might be turned off or on at any time, so the set V of alive nodes varies with time and has no fixed size.

Finally, we assume an abstract entity called an *overlay*, which is simply a collection of nodes. Nodes that belong to the overlay are called *overlay backbone nodes*. Part of what we do in this paper is investigating various protocols for deciding which nodes should be in the overlay.

Caches allow a set $V^t(d) \subseteq V$ of nodes to access a data d at time t . For a given node p , we define $V_p^t(d) \subseteq V^t(d)$ to be the subset of nodes $q \in V^t(d)$ such that there is a path between p and q at time t . For a node p requesting a data d , each node $q \in V_p^t(d)$ is an *alternative* for retrieving d .

A lookup service is initiated by a *request*, which is characterized by a requester $p \in V$, a data item d , and a time t at which the request is initiated. In response to a request, the lookup service is supposed to generate a *reply*, which consists of at least one alternative, if one exists, or \perp .

We restrict the allowed behavior of the lookup service by requiring it to provide *validity* and *effectiveness*. The validity property specifies that a lookup service result is not spurious while the effectiveness property specifies the ability of a lookup service to produce a result. Moreover, as we would often like to also limit the time spent by the lookup service on each request, we propose two definitions of effectiveness: *timed effectiveness* and *eventual effectiveness*.

Validity: If a lookup reply for a request (p, d, t) is generated at time t' and includes node q , then there exists a time t'' such that $t \leq t'' \leq t'$ and $q \in V_p^{t''}(d)$.

Timed Effectiveness: There is a time $T > 0$ such that for every request (p, d, t) for which there exists an alternative at time t' such that $t \leq t' \leq t + T$, the service generates a reply with an alternative at time t'' such that $t \leq t'' \leq t + T$.

Eventually Effectiveness: For every request (p, d, t) for which an alternative exists at some time $t' \geq t$, the service eventually generates a reply with an alternative.

Note that due to node mobility and the possibility of evacuating a data item from the cache, validity only ensures that a corresponding data item existed at the alternative when the reply was generated. However, it is possible that by the time the reply is received by the requesting node, the data no longer exist at this alternative. Similarly, it is possible that by the time a request is propagated, the alternative is either no longer reachable, or the data object has been removed from the alternative's cache. Additionally, due to mobility, ensuring effectiveness always might be very costly, and practically sometimes impossible. Thus, we strive to ensure effectiveness with high probability rather than definitively.

3. IMPLEMENTING LOOKUP SERVICES

3.1 (Constrained) Flooding

The *flooding* scheme is the simplest way to disseminate a request in a network [14]. A requester sends a *search* message to all its neighbors in the network. When a node receives a request for the first time, if it does not have the data, it forwards the request to all its neighbors while recording the path the request has traveled so far on the forwarded message. If a node that receives a request has a copy of the data, it immediately sends a response along the reverse path and does not forward the request any further. Nodes also discard redundant copies of the same request.

Alternatively, when an overlay exist, then only nodes that are part of the overlay backbone forward the message to their overlay neighbors. Other nodes only reply if they have a copy of the data and drop the request otherwise. We refer to running flooding on top of an overlay as *constrained flooding*.

Among the advantages of flooding, we note that the time latency is minimal. Also, this is the most robust scheme and has the highest probability of reaching all network nodes, and therefore find an alternative if one exist. However, the price for this is that unconstrained flooding generates a large number of messages. This means that the energy consumption of flooding is very high, and so is the bandwidth utilization. Since the network is shared between the lookup service and the application, this comes at the expense of the bandwidth left for application messages. Also, this increases the chances of collisions, which could degrade the overall performance of the system.

There are two common ways to limit this bad behavior of flooding. One option is to limit flooding with a *Time-To-Leave* (TTL) parameter, which specifies the maximum number of hops a search request can be forwarded on. This technique solves some scalability problems [10], but is still not very efficient. Alternatively, constrained flooding greatly reduces the number of messages sent on each search request, since only the overlay backbone nodes participate in the forwarding. However, this also imposes several challenges, since computing the overlay backbone also requires some resources, and one needs to be careful to balance the cost of the overlay maintenance with the benefits it brings. This is particularly true in MANETs due to their dynamic nature, which means that the overlay is continuously evolving.

3.2 Probabilistic Flooding

Probabilistic flooding is similar to constrained flooding, except that nodes only forward incoming messages with some probability. In dense networks, when a host decides to forward a message, all of its neighbors might already receive the message. Deciding randomly that some nodes would not forward a message can save unnecessary messages without harming effectiveness. But, in sparse networks, some nodes will not receive all the messages unless the probability of rebroadcasting a message is high.

One solution is to adapt the probability parameter to the number of neighbors. Another idea is to decide whether to forward a message based on the number of redundant messages a host receives [17, 25]. That is, each node counts the number of times it received each message during a given time interval starting from the arrival of the first copy of the message. At the end of this time interval, if the number is less than a threshold value, the message is forwarded, and it is dropped otherwise. This way, in dense networks, some nodes will not rebroadcast messages, while all nodes rebroadcast in sparse network. However, in this solution messages are delayed at each hop, which greatly increases the delivery latency.

We propose a novel hybrid solution that consists of determining the probability parameter based on the average number of redundant messages measured during a longer period of time. We note \bar{r} the average number of redundant messages measured during a fixed period of time, R is the ideal number of redundant messages a host should receive, and p is the probability parameter. At the beginning, p is set to some initial value.

On each message receipt, each node recomputes its probability parameter to be $p \leftarrow (1 + d) * p$ where $d = k * (R - \bar{r})$ (naturally, this imposes a constraint that $k * (R - \bar{r}) \leq (1 - p)/p$). In this function, k is a parameter that permits quick or slow evolution of the probability parameter. If \bar{r} is less than R , it means that the node does not receive enough redundant messages. In this case, $R - \bar{r}$ is positive, so p grows. Otherwise, $R - \bar{r}$ is negative and thus p decreases.

In our experiments, reported in Section 5, we set $R = 5$ and $k = 0.05$. The value of R is based on the work of [17, 25], in which they investigated the optimal value of such a parameter under both fixed and dynamic methods, but for a deterministic forwarding scheme. The value of k was set in an experimental manner.

The main disadvantage of this protocol is that in lookup services the number of messages depends on the number of requests submitted to the service, so this protocol is only effective when there are many messages. Thus, when there are few requests, the accuracy of the computation is poor, and it does not ensure effectiveness.

3.3 BFS

Another scheme is based on the Breadth First Search algorithm [6]. This algorithm can be viewed as successive instantiations of flooding with increasing TTL values ranging from 1 to the expected diameter of the network. More specifically, a requester $p \in V$ first initiates the lookup by sending a request to all nodes in $\mathcal{N}(p)$. Upon receiving a request, a node responds with a positive message if the requested data is in its cache and by a negative message otherwise. If the requester does not receive any positive response, it sends another request to all nodes in $\mathcal{N}^2(p)$. Nodes that did not receive the message during the first round send their response along the reverse path. This process continues iteratively until either the requester receives a positive answer, or the requester believes it has reached all nodes.

In this protocol, the message propagation is controlled by the requester. Once an alternative is detected, the propagation stops, which saves a substantial number of unnecessary messages when there exists an alternative close to the requester. Note that this idea can be applied over an unstructured network, as been described above, or using an overlay. In the latter case, only the overlay backbone nodes forward the request, while preserving the TTL of the corresponding iteration.

The main drawback of this protocol is its potential long latency, as messages are not propagated to nodes at distance i hops until a timeout after which it is assumed that all replies from nodes

at distance $i - 1$ have been received. Moreover, it may generate many messages when the proxy is far, since search messages travel repeatedly through nearby nodes in successive invocations of the search.

4. OVERLAY MAINTENANCE

4.1 A Maximal Independent Set (MIS) Based Overlay

We start this section by giving a formal definition of a *Maximal Independent Set* (MIS) and then discuss how to obtain an MIS based overlay.²

Let $G = (V, E)$ be a communication graph. Two nodes i and j in G are said to be *independent* if $(i, j) \notin E$. A subset $S \subseteq V$ of nodes is *independent* if every pair of nodes in S are independent. A set S is a *maximal independent set* (MIS) if S is independent, yet for any node $k \in V \setminus S$, $S \cup \{k\}$ is not independent.

The MIS based overlay is constructed in two phases that are executed in parallel. In the first phase, the MIS is computed. Since by definition, the set of nodes in an MIS cannot directly communicate with each other, the second phase identifies bridge nodes that connect the MIS nodes. Of course, the goal is to find as few bridge nodes as possible, yet to do this in a completely decentralized manner.

So, we are interested in a distributed algorithm for computing an MIS in such a way that every node makes local calculations based only on the knowledge of its neighbors. Recall that the neighbors of p are the nodes that appear in the transmission disk of p , and thus p can communicate directly with them, and every message p sends is received by all of them. Additionally, we would like to influence the overlay construction process such that the overlay nodes will be the “best” nodes under a given metric. For example, since in mobile systems nodes are often battery operated, we may wish to use the energy level as the metric, in order to have the nodes with highest energy levels members of the overlay. Alternatively, we might use the number of objects for which a node is proxy as the metric, in order to reduce the average number of hops a search message has to travel. Similarly, we might use bandwidth, transmission range, or local storage capacity, or some combination of several such metrics.

This is achieved by having a generic function which associates with each node some value from an ordered domain, which represents the node’s appropriateness to serve in the overlay. We call this value the *goodness number*. This way, it is possible to compare any two nodes using their goodness number and to prefer to elect the one whose value is higher to the overlay. For example, it is easy to evaluate and compare the battery level of nodes, or to obtain and compare the number of objects for which a node is proxy.

The MIS algorithm consists of computation steps that are taken periodically and repeatedly by each node. In each computation step, each node makes a local computation about whether it thinks it should be in the MIS or not, and then exchanges its local information with its neighbors. For simplicity, we concentrate below on the local computation steps only.

The local state of each node includes a *status*, which is either *active*, *passive*, or *bridge*, its goodness number, and its knowledge of the local states of all its neighbors (based on the last local state they reported to it), and for each neighbor, the list of its active neighbors. The *active* status means that the node believes it is in the

²Note that we are interested in a maximal independent set and *not* the maximum independent set; the latter is a hard problem whereas the former is solvable.

MIS, *bridge* means that it is acting as a bridge, and *passive* means that it is neither.

The local execution of the MIS part of the protocol includes the rules \mathcal{R}_1 , \mathcal{R}_2 , and \mathcal{R}_3 of Module 1. Our MIS construction is more efficient than the one proposed for sensor networks in [11] since our algorithm dynamically recomputes the MIS function based on the goodness number. The first rule, \mathcal{R}_1 , is used to elect nodes that have the maximal goodness number in their neighborhood to the MIS. The second rule, \mathcal{R}_2 , is used to ensure that we do not have a situation in which two neighboring nodes are in the MIS. This could happen, for example, due to movement of nodes, which changes their neighborhood. Thus, if an active node i finds that one of its neighbors is active and has higher goodness number, then i gives up and becomes *passive*. The third rule, \mathcal{R}_3 , is executed by *passive* and *bridge* nodes that do not have an *active* neighbor even though they do not have the highest goodness number. This situation can occur, for example, if all neighbors of i that have higher goodness number than i gave up being *active* due to the fact that they had other neighbors with even higher goodness numbers.

Rules \mathcal{R}_1 and \mathcal{R}_3 are based on the evaluation of the maximal goodness number in a neighborhood. In case two nodes have the same goodness number, the symmetry is broken using ids. In the following, e_i denotes the goodness number of node i . In order to simplify the algorithms presentation we introduce the goodness relation denoted in the following by \prec . Node j is better than node i according to the \prec relation if either the goodness number of i is superior to the goodness number of j or the nodes have the same goodness number but the identifier of j is greater than the identifier of i . Formally, $i \prec j$ iff $e_i < e_j \vee e_i = e_j \wedge id_i < id_j$. Note that \prec defines a total order on the nodes when the goodness values are comparable. Also, $MAXE(i)$ is a boolean predicate that evaluates to true if and only if i is maximal in its neighborhood w.r.t. the relation \prec defined above.

Module 1 Goodness-based MIS executed by node i

Input Parameters:

$\mathcal{N}(i)$: set of i 's neighbors (including i itself);
 $status_j, \forall j \in \mathcal{N}(i), j \neq i$: the status of each i neighbor;
 $e_j, \forall j \in \mathcal{N}(i)$: the goodness number of neighbor j ;

InOut Parameters: $status_i$: the status of node i

Predicates: $i \prec j \equiv e_i < e_j \vee e_i = e_j \wedge id_i < id_j$
 $MAXE(i) \equiv \forall j \in \mathcal{N}(i), j \neq i, j \prec i$

Actions:

\mathcal{R}_1 : $MAXE(i) \wedge status_i \neq active \rightarrow status_i := active$
 \mathcal{R}_2 : $status_i = active \wedge \exists j \in \mathcal{N}(i), status_j = active \wedge i \prec j \rightarrow status_i := passive$
 \mathcal{R}_3 : $\neg MAXE(i) \wedge \forall j \in \mathcal{N}(i), status_j \neq active \rightarrow status_i := active$

The algorithm shown as Module 2 provides the bridges between the MIS nodes (recall that MIS nodes are not connected to each other). Module 2 has three rules and shares with Module 1 the set of neighbors, the status variables, and the goodness numbers. The first rule, \mathcal{R}_1 , is used to connect between two nodes in the MIS. Specifically, if node i is *passive*, and has two neighbors which are *active*, then i becomes a *bridge*. The second rule, \mathcal{R}_2 , is used to eliminate useless bridges that are no longer connecting any two distinct *active* members. Such bridge nodes switch back to being *passive*. Finally, the rule \mathcal{R}_3 is used to eliminate redundant *bridge* nodes. That is, whenever there are two neighboring *bridge* nodes i and j such that the set of *active* neighbors of i is included in the set of j , or both have the same *active* neighbors but i has lower

goodness number than j , then i switches back to *passive*.

Module 2 Bridge Construction executed by node i

Input Parameters:

$\mathcal{N}(i)$: set of i 's neighbors;
 $AN(i)$: set of i 's active neighbors;
 $status_j, \forall j \in \mathcal{N}(i), j \neq i$: the status of each j neighbor;
 $e_j, \forall j \in \mathcal{N}(i)$: the goodness number of neighbor j ;

InOut Parameters: $status_i$: the status of i

Predicates: $i \prec j \equiv e_i < e_j \vee e_i = e_j \wedge id_i < id_j$

Actions:

\mathcal{R}_1 : $status_i = passive \wedge \exists j, k \in \mathcal{N}(i), j \neq k, status_j = status_k = active \rightarrow status_i := bridge$
 \mathcal{R}_2 : $status_i = bridge \wedge \exists j, k \in \mathcal{N}(i), j \neq k, status_j = status_k = active \rightarrow status_i := passive$
 \mathcal{R}_3 : $status_i = bridge \wedge \exists j \in \mathcal{N}(i), j \neq i, status_j = bridge \wedge (AN(i) \subset AN(j) \vee (AN(i) = AN(j) \wedge i \prec j)) \rightarrow status_i := passive$

4.2 A Connected Dominating Set (DS) Based Overlay

We start this section by giving a formal definition of a *Connected Dominating Set* (DS), and then discuss how to obtain a DS based overlay.

Let $G = (V, E)$ be a communication graph. A set $S \subset V$ is a *dominating set* if any node in V is a member of S or has a neighbor in S . S is *connected* if for any node x in S there is another node y in S such that $(x, y) \in E$.

In the following, we present a self-stabilizing version of the DS-algorithm construction of [28, 29]. The proactive DS algorithm (Module 3) requires each node to know about its neighbors at distance two, or in other words, the neighbors of its direct neighbors. The protocol also uses the neighbors independence predicate, as defined below:

Definition 4.1 (independent neighbors) Let $G = (V, E)$ be the communication graph and let $i \in V$. $Independent_Neighbors(i) \equiv \exists y, k \in \mathcal{N}(i), y \neq k \neq i, k \notin \mathcal{N}(y) \wedge y \notin \mathcal{N}(k)$.

Intuitively, the predicate $Independent_Neighbors(i)$ evaluates to true if there are two neighbors of i , y and k , that are not direct neighbors of each other. If this predicate is true, then i should be in the dominating set, unless there is another node j that is a neighbor of both y and k and has a higher goodness number.

Thus, a node i executes the rule \mathcal{R}_1 if the predicate $Independent_Neighbors$ is true for i , and becomes *active*. However, by the rule \mathcal{R}_2 , if an *active* node i finds another *active* neighbor and both share the same neighbors, yet the other node has a higher goodness number, then i gives up. Note that at the beginning of the protocol all nodes might be *passive*. Also, if the graph is fully connected (a clique), then no node will become *active* by \mathcal{R}_1 . This is taken care of by the rule \mathcal{R}_3 , in which if all the nodes in i 's neighborhood are *passive* and i has the maximal goodness number, then i becomes *active*. The last rule, \mathcal{R}_4 , eliminates redundant *active* nodes; if a node i is *active*, but it has another *active* neighbor j and the set of i 's neighbors is included in j 's neighbors, then i gives up and becomes *passive*.

Module 3 Goodness based DS executed by node i

Input Parameters:

$\mathcal{N}(i)$: set of i 's neighbors;
 $status_j, \forall j \in \mathcal{N}(i), j \neq i$: the color of each i neighbor;
 $e_j, \forall j \in \mathcal{N}(i)$: the goodness number of neighbor j ;
 $\mathcal{N}(j), \forall j \in \mathcal{N}(i)$: the set of neighbors of neighbor j of i ;

InOut Parameter: $status_i$: the status of node i

Predicates:

$Independent_Neighbors(i) \equiv \exists y, k \in \mathcal{N}(i), y \neq k \neq i, k \notin \mathcal{N}(y) \wedge y \notin \mathcal{N}(k)$
 $i \prec j \equiv e_i < e_j \vee e_i = e_j \wedge id_i < id_j$
 $MAXE(i) \equiv \forall j \in \mathcal{N}(i), j \neq i, j \prec i$

Actions:

$\mathcal{R}_1: status_i = passive \wedge Independent_Neighbors(i) \rightarrow status_i := active$

$\mathcal{R}_2: \exists j \in \mathcal{N}(i), \mathcal{N}(j) = \mathcal{N}(i) \wedge i \prec j \wedge status_i = status_j = active \rightarrow color_i := passive$

$\mathcal{R}_3: \forall j \in \mathcal{N}(i), \mathcal{N}(j) = \mathcal{N}(i) \wedge MAXE(i) \wedge status_i = passive \rightarrow status_i := active$

$\mathcal{R}_4: \exists j \in \mathcal{N}(i), \mathcal{N}(i) \subset \mathcal{N}(j) \wedge status_i = status_j = active \rightarrow status_i := passive$

4.3 BFS Tree Protocol

The idea in this protocol is to exploit the synchronous behavior of BFS and the broadcast nature of wireless networks to produce an efficient reactive dissemination tree for each search request. Interestingly, this is obtained without any additional delay or control messages. Specifically, recall that with BFS dissemination, the search propagates in synchronous rounds such that in round i the search messages disseminate until distance i hops from the requester. The requester waits for receiving the reply messages of round i before continuing to round $i + 1$. Thus, when the requester receives all replies for round i , it knows the fastest paths between all nodes at distance i and itself. Based on this, it can calculate the minimal set of nodes that need to forward the search during the next round and ensure that the message will still reach all nodes at distance $i + 1$.

To obtain this, the requester specifies in the request message the overlay backbone nodes. Thus, the request message is defined by $\langle \mathbf{request}, \mathbf{requester}, \mathbf{data}, \mathbf{round}, \mathbf{dist}, \mathcal{R}, \mathcal{P} \rangle$ where $\mathbf{requester}$ is the requester, \mathbf{data} is the requested data, \mathbf{round} is the distance of the nodes the requester wants to reach, \mathbf{dist} is the number of hops the search has traveled so far, \mathcal{R} is the set of nodes that should forward the search message and \mathcal{P} is the set of nodes on the path of the message has traveled. A node forwards an incoming request message if its hop-distance to the requester is less than \mathbf{round} and if it belongs to R . In this case, it also adds itself to \mathcal{P} .

5. SIMULATIONS

5.1 Simulation Environment

We assume a simulation area of size $500 \times 500 \text{ m}^2$. Nodes are initially placed in random locations in the simulation area. Then, during the simulations, nodes move according to an Augmented Random-Waypoint model. That is, according to the Random - Waypoint model [13], each node picks a new random target location and moves there with a random velocity (under some pre-defined maximum). Once the node gets to its target location, it remains there for a random amount of time, which in our simulations ranged between 1 to 2 minutes.

Yet, when considering practical environments, it is likely that nodes will often not move to completely random locations, but rather

Module 4 Tree BFS Protocol – code for node p

Parameters:

$\mathcal{N}(i)$: set of i 's neighbors
 $construct_tree(\Lambda)$: return a minimal tree from the set of paths Λ

Actions:

\mathcal{R}_1 : node i initiates a lookup

$round := 1; \mathcal{R} := \emptyset;$

repeat:

send $\langle \mathbf{request}, i, \mathbf{DATA}, \mathbf{round}, 1, \mathcal{R}, \emptyset \rangle$

wait $\langle \mathbf{response}, j, \mathbf{presence}_j, \mathcal{P}_j \rangle$ from all

$j \in \mathcal{N}(i) \cap \mathcal{R}$ or timeout

if received at least one $\langle \mathbf{response}, j, \mathbf{true}, \mathcal{P}_j \rangle$ then

return the smallest j for which such a message was rec.

elseif received no response then return \perp

endif

$round := round + 1; \mathcal{R} = construct_tree(\bigcup_j \mathcal{P}_j)$

end repeat

\mathcal{R}_2 : upon the first rec. of $\langle \mathbf{request}, j, \mathbf{DATA}, \mathbf{round}, \mathbf{dist}, \mathcal{R}, \mathcal{P} \rangle$

if $\mathbf{dist} = \mathbf{round}$ then send $\langle \mathbf{response}, i, \mathbf{true}, \mathcal{P} \cup \{i\} \rangle$

if \mathbf{DATA} is present at i then

else send $\langle \mathbf{response}, i, \mathbf{false}, \mathcal{P} \cup \{i\} \rangle$

endif

else

if $i \in \mathcal{R}$ then

send $\langle \mathbf{request}, i, \mathbf{DATA}, \mathbf{round}, \mathbf{dist} + 1, \mathcal{R}, \mathcal{P} \cup \{i\} \rangle$

endif

endif

will be attracted by a few *hot-spots*. For example, in a trade-show, participants wander between booths. Along this line, we have augmented the Random-Waypoint model to include privileged subareas, namely the hot-spots, such that each time a node chooses a new position it prefers a position inside a hot-spot with a given variable probability. By varying this probability, we modify the density homogeneity of the network. When this probability is 0, the density of the simulation area is uniform. As this probability increases, hot-spots become more dense at the expense of the rest of the simulation area. In our simulations, we have defined three square hot-spots of sizes $50 \times 50 \text{ m}^2$, $62 \times 62 \text{ m}^2$, and $83 \times 83 \text{ m}^2$.

Parameters	values	Parameters	values
Nb. nodes	200	Transm. range	70 m
Node velocity	2 m/sec.	Request period	700 msec.
Transm. time	[1 . . . 4] msec.	Nb. proxies	8
Hot-spots prob.	0.6	Packet loss	no

Table 1: Default Parameters for Simulations

The simulation duration is 8 minutes. For the MIS and DS implementations, the frequency of node status re-computation is uniformly chosen between 2 and 3 seconds. Table 1 lists the default values of the parameters we use for this set of simulations. In our simulations, we have varied the node velocity, hot-spot probability, transmission range, and ratio of proxies. The performance metrics we measured are the *number of messages by request*, the *time latency* and the *hit ratio*. Moreover, we have studied the following schemes: *flooding* – unconstrained flooding, *MIS flooding* – MIS based constrained flooding, *DS flooding* – DS based constrained flooding, *probabilistic* – probabilistic flooding, *BFS-tree* – BFS along the BFS-tree scheme, *MIS-BFS* – BFS along the arcs of the MIS overlay.

5.2 Hit-Ratio Comparison

Figure 1 presents the hit-ratio of all studied schemes when varying the number of proxies in the system. As can be expected, the hit-ratio increases as the number of proxies increase and approaches one. Yet, the increase has a knee between 5 and 10 proxies, or in other words, between 2.5% and 5% of the nodes. This result is significant, as it indicates an optimal replication degree of cache proxies in order to ensure that at least one proxy is found with very high probability. Obviously, unconstrained flooding exhibits the highest hit-ratio, although with 5 proxies, the BFS-tree scheme and the dominating set scheme match the performance of unconstrained flooding. This is interesting, since the BFS-tree scheme is completely reactive. Also, probabilistic flooding fares quite well when the number of proxies is above 10. The significance of this is that this is the simplest and cheapest scheme!

Figure 2 presents the obtained hit-ratio when varying the transmission range. Here we see a similar pattern as before, yet the knee is less dramatic. Yet, we would like to point out the relatively good behavior of probabilistic flooding when the transmission range is short (or when a network is sparse). However, as the transmission range increases, the performance of probabilistic flooding improves more slowly than for other schemes.

Figure 3 explores the hit-ratio vs. the nodes maximal velocities. Here, we can see that increased speed reduces the hit ratio. This is due to the fact that as node move fast, the chance that a search message will get lost is higher, and similarly, the chance that a return path for the response will become invalid is higher. Moreover, the MIS based schemes suffer more significant performance degradation than others, as the MIS overlay does not keep up with the changes. Interestingly, DS fares better than MIS. The reason is that our DS scheme had more redundancy than the MIS scheme, and thus is more vulnerable to quick changes.

Finally, Figure 4 studies the hit-ratio vs. the hot-spot probability. It appears that there is a slight drop in hit-ratios as this probability increases. This can be explained by the fact that as nodes concentrate in hot-spots, there is a greater chance of partitions, and thus some searches cannot find their proxies. Here too, the MIS based schemes were the most vulnerable. This is due to the fact that the overlay they create include the fewest number of nodes. On the other hand, the DS scheme performed the best. This is due to its particular design, which in the case of hot-spots, has a good chance of establishing an overlay that includes nodes that connect between the hot-spots.

5.3 Latency Comparison

Figure 5 exhibits the latency of all studied schemes vs. the number of proxies. As expected, the latency drops as the number of proxies increases, since it also increases the probability of having a nearby proxy. Interestingly, here too we see a sharp knee between 5-10 proxies, or between 2.5%-5%, which strengthen the corresponding observation regarding the optimality of the replication degree of proxies. Also, BFS based schemes require much longer time to generate a response, due to their synchronous iterative style.

Figure 6 presents the latency vs. the transmission range. As it appears, flooding based schemes are hardly impacted by the range, although do exhibit some improvement as the range increases. On the other hand, for BFS based schemes, the improvement is much more significant, since the longer transmission range reduces the number of iterations needed to find a proxy, and each iteration takes a significant time.

Figure 7 studies the observed latency vs. nodes maximal velocities. It appears that nodes velocities have little impact on the laten-

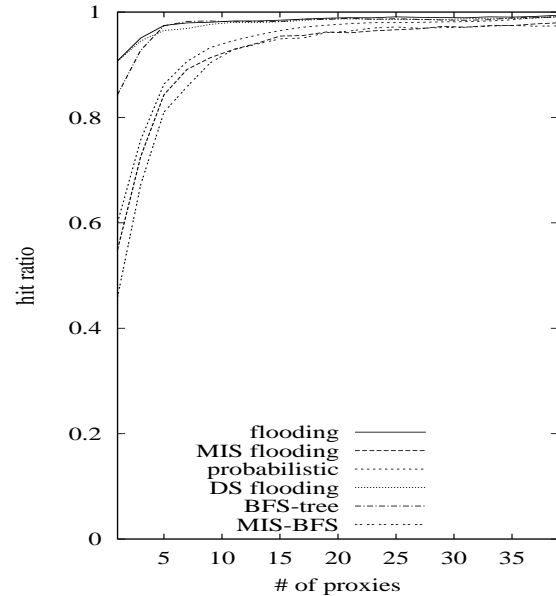


Figure 1: Hit-ratio vs. number of proxies

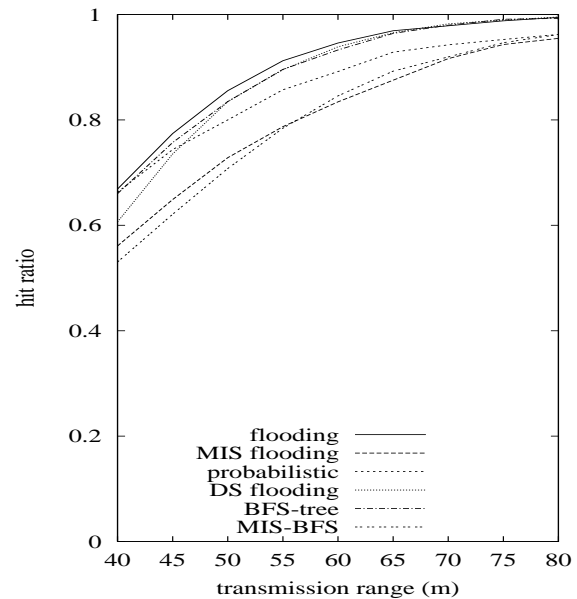


Figure 2: Hit-ratio vs. transmission range

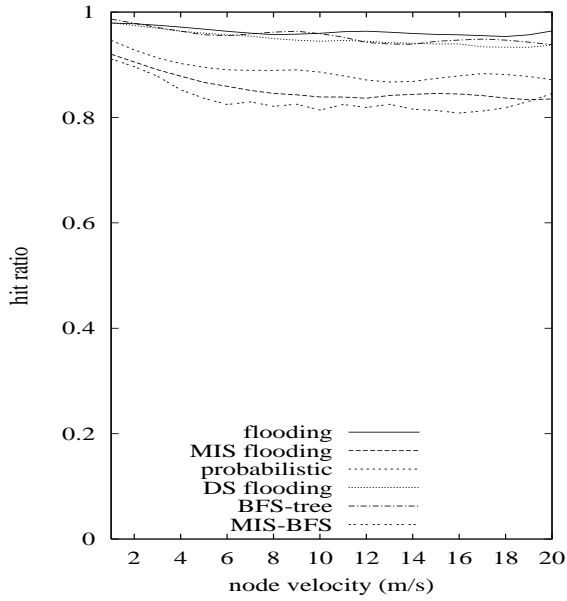


Figure 3: Hit-ratio vs. node velocity

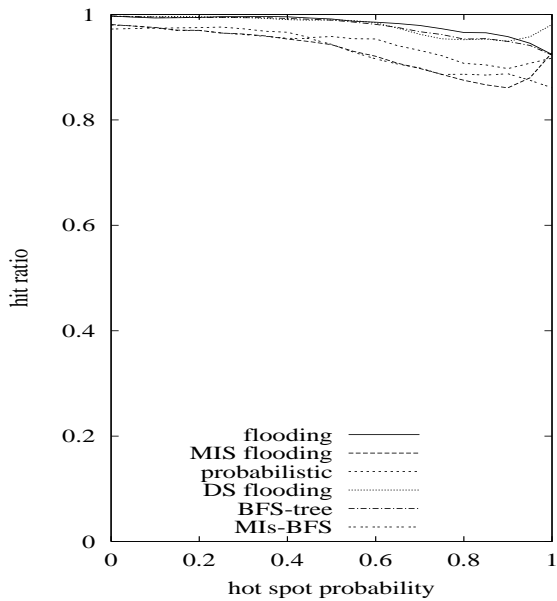


Figure 4: Hit-ratio vs. hot-spot probability

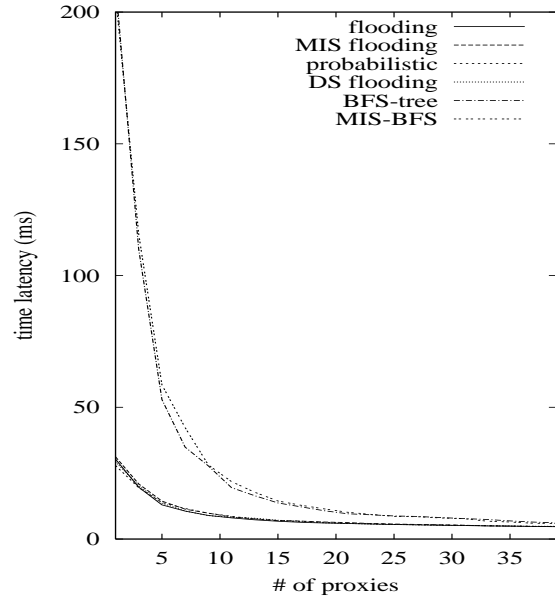


Figure 5: Latency vs. number of proxies

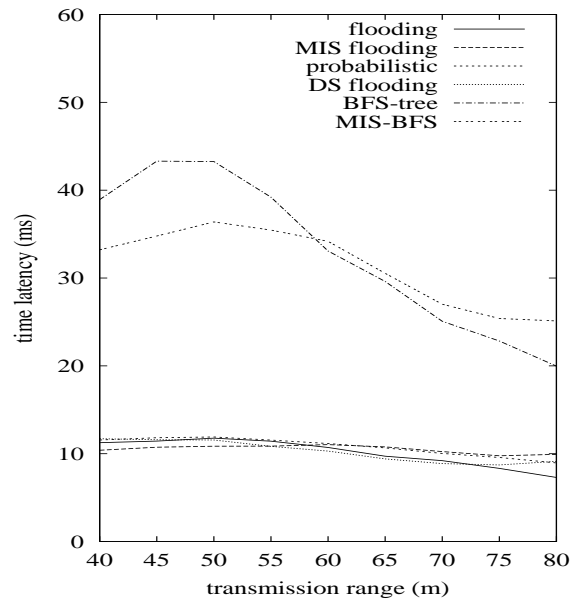


Figure 6: Latency vs. transmission range

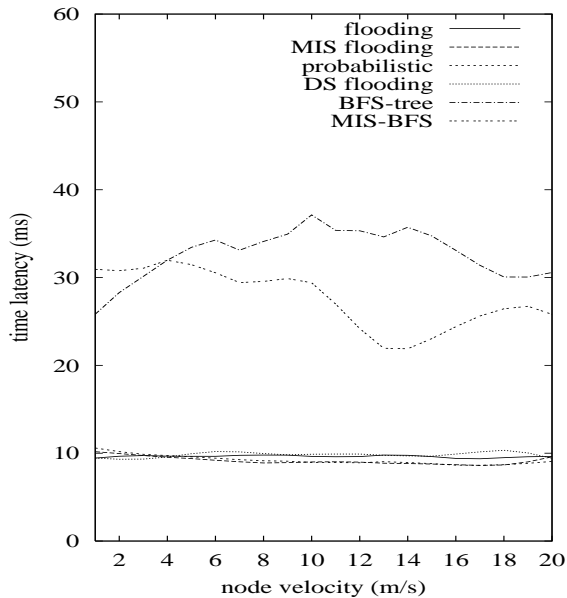


Figure 7: Latency vs. node velocity

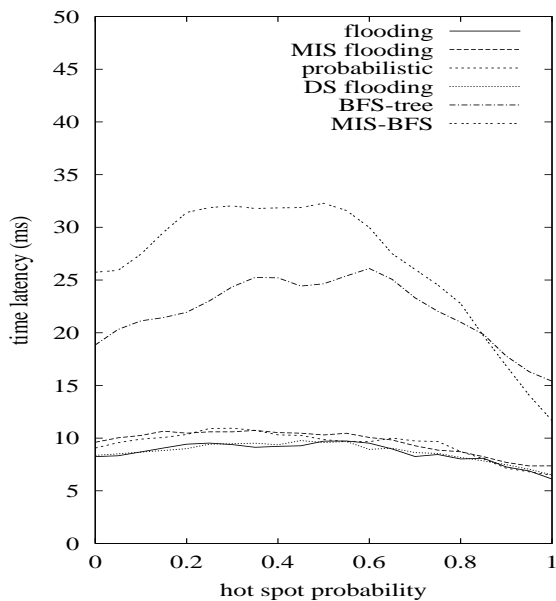


Figure 8: Latency vs. hot-spot probability

cy. Finally, Figure 8 explores the latency vs. hot-spot probability. Here there is a large improvement for the BFS based schemes, and minor improvement for the other schemes. The reason is again that when nodes are in condensed areas, if a proxy is found, it is nearby, and thus is found with fewer iterations, and each iteration takes a long time.

5.4 Communication Requirements Comparison

Figure 10 presents the number of messages generated per request by each studied scheme vs. the number of proxies. Here we can see a very interesting phenomenon. On one hand, the communication cost of flooding based schemes grows proportionally to the number of proxies. On the other hand, with BFS based approaches, it is the opposite. In particular, with 10 proxies, or 5% of nodes, the BFS schemes induce fewer messages than the others. This can be explained by the fact that when there are many proxies, there is a good chance that one exists nearby. As BFS stops the search process as soon as the first proxy is found, it generates fewer messages in these cases. As for the flooding based schemes, they have no way of pruning search messages that do not pass through a proxy, and thus find multiple proxies, each of which generates a response. Among the flooding schemes, unconstrained flooding is clearly by far the worst and MIS is by far the best. This indicates that MIS generates the most efficient overlay among the schemes we studied, i.e., the overlay with fewest nodes. Moreover, the BFS-MIS scheme is even better than the BFS-tree, as it limits the nodes on which the BFS proceeds (this is true for all simulations in this subsection).

Figure 10 investigates the number of messages generated by a request vs. the transmission range. Here we see an increase in the number of messages when the transmission range increases. This is because with longer transmission ranges, each message is received by a larger number of nodes, each of which, if it is not a proxy, forwards it.

Figure 11 explores the number of messages per request vs. nodes maximal velocities. Here we can see that the latency reduces with the increase in speed. Yet, this seems to be, in fact, a negative result, as it is caused by the fact that with fast movement, more search messages are lost, and thus their corresponding searches are pruned earlier.

Figure 12 studies the number of messages per request vs. hot-spot probability. In most schemes there is a dramatic drop in the number of messages as this probability increases. This is due to the fact that when all nodes are in hot-spots, either a proxy is found inside the hot-spot, or it will probably not reach the other hot-spots. As a result, much fewer messages are sent around, but as mentioned earlier, this comes at the expense of reduced hit-ratio. The BFS based schemes were less influenced by this since the other simulation parameters are such that even without hot-spots they already obtain relatively good results.

6. DISCUSSION

In this paper we have outlined several search techniques for locating cache proxies. These include flooding (constrained and unconstrained), a novel variant of probabilistic flooding, and BFS. We have also examined several overlays on which these search methods can be applied, including a new construction for MIS based overlays, a DS based overlay, and a BFS-tree overlay. All our constructions are self-stabilizing and therefore react automatically to dynamic changes in the environment. We have also studied the performance of several meaningful combinations of the search techniques and overlay maintenance schemes by extensive simulations.

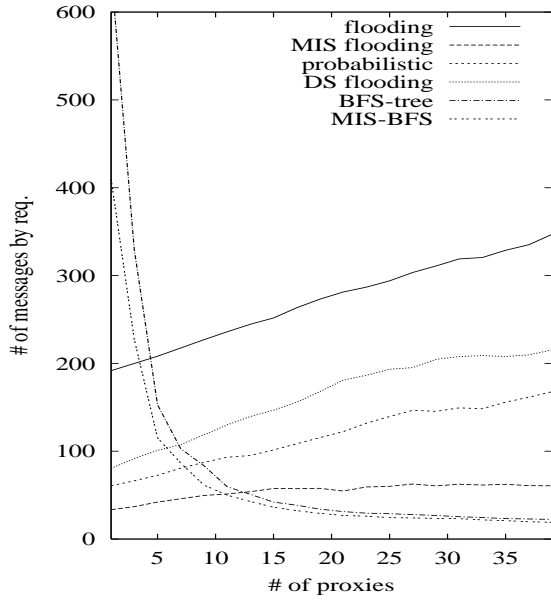


Figure 9: Number of messages vs. number of proxies

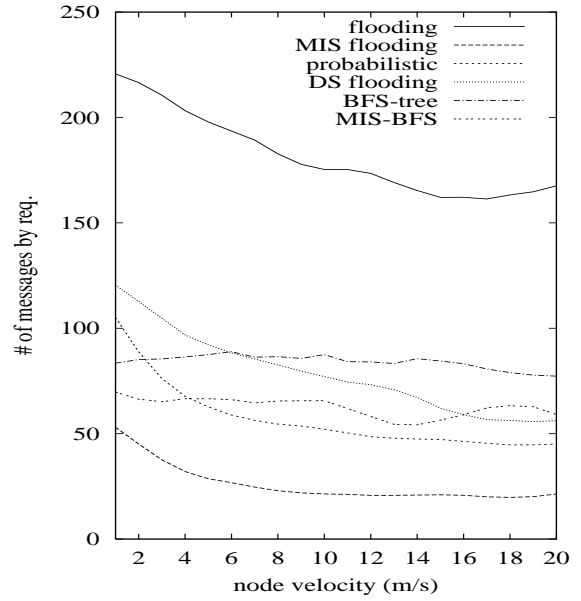


Figure 11: Number of messages vs. node velocity

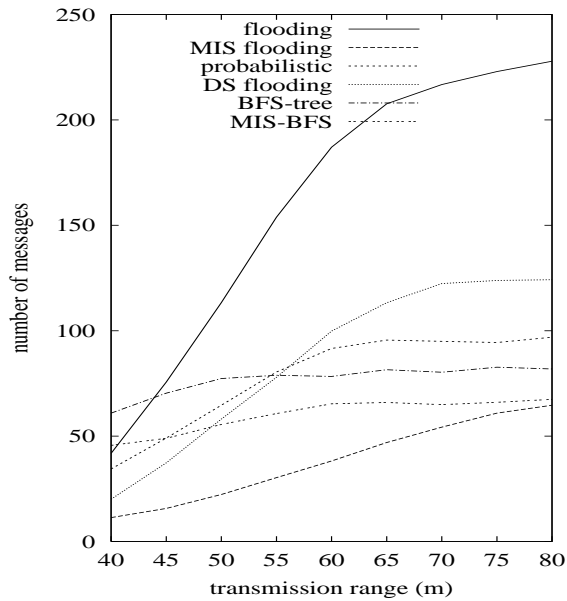


Figure 10: Number of messages vs. transmission range

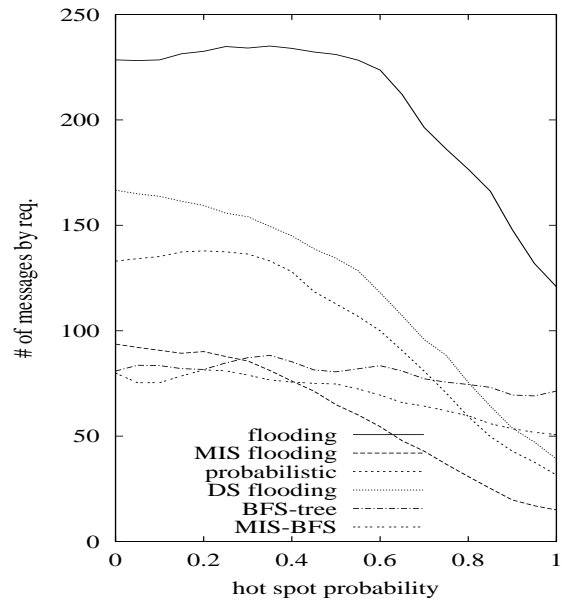


Figure 12: Number of messages vs. hot-spot probability

Our simulations have discovered the following phenomena: In general, as can be expected, there is a tradeoff between efficiency and effectiveness. That is, the more efficient a scheme is, the worse it performs in terms of hit-ratio. Interestingly, we have also found that there exists an optimal number of proxies for a data item, in our case 2.5%-5% of the nodes, which ensures that a proxy will be found with very high probability. Adding proxies for the same data item beyond this number will hardly improve the hit-ratio. Similarly, the same number of proxies is also optimal for lookup latency. This result seems significant for proxy placement designers.

Surprisingly, BFS techniques performed better in terms of efficiency when the number of proxies is optimal or higher. This is due to their ability to prune the search once a proxy is found. Thus, when the number of proxies is expected to be high, this search technique becomes very attractive, as it is also completely reactive, although in terms of latency it was still somewhat slower than other schemes. On a similar note, the probabilistic flooding scheme performed reasonably well under most metrics. Again, this makes it attractive, as it is the simplest and cheapest to implement.

Overall, the proactive overlay based schemes exhibited disappointing results. Moreover, in this work we did not count the cost of the messages required to maintain the overlay. When adding this cost, it seems that, at least for the schemes we have studied, the cost of reactive overlay is not worth the benefits. On the other hand, in quasi-static ad-hoc environments, such as sensor networks, it may still be useful to employ reactive overlays. This is because in such systems the topology rarely changes and, thus, the cost of maintenance can be lower and the observed performance better.

One immediate future extension of our work is adding A* like pruning capabilities to the various search techniques.³ That is, the search messages carry with them a field that measures the cost of the search so far. In addition, at each node, one assumes the existence of an estimated cost function for continuing the search along that path. Once the combined measured cost and estimated additional costs are above a given threshold, the search is pruned there.

Another extension to our work can be to look for other reactive and quasi-reactive search schemes. In particular, in purely reactive schemes, nodes are not allowed to generate their own messages. Thus, nodes only learn of neighbors when a search is propagating through these neighbors. It is interesting to consider quasi-reactive schemes of overlay based approaches, in which the overlay is built based on the perceived neighborhoods. Yet, nodes are allowed to generate their own messages only when they suspect that the information they have is too old to the point that the system's perceived performance degrades below a given threshold.

7. REFERENCES

- [1] M. Abrams, C. R. Standridge, G. Abdulla, S. Williams, and E. A. Fox. Caching proxies: Limitations and potentials. In *Proceedings of the 4th International WWW Conference*, December 1995.
- [2] K. Alzoubi, Wan Peng-Jun, and Ophir Frieder. Weakly connected dominating sets and sparse spanners in wireless adhoc networks. *Proc. of the 23th International Conference on Distributed Computing Systems (ICDCS'03)*, pages 96–104, 2003.
- [3] C.M. Bowman, P.B. Danzig, and D. Hardy. The harvest information discovery and access system. *Computer Networks*, 28(1-2), 1995.

³In fact, A* was originally introduced as an extension of Depth First Search (DFS). DFS is inherently centralized, but the basic idea of heuristic pruning can be applied to any search strategy.

- [4] G. Chockler, D. Dolev, R. Friedman, and R. Vitenberg. Implementing a caching service for distributed corba objects. In *Proc. Middleware 2000: IFIP/ACM International Conference on Distributed Systems Platforms*, pages 1–23, April 2000.
- [5] T. Clause, P. Jacquet, and A. Laouti. Optimized link state routing protocol. *INMIC'01, Pakistan*, 2001.
- [6] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms (Second Edition)*. MIT Press and McGraw-Hill, July 2001.
- [7] E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, November 1974.
- [8] L. Fan, P. Cao, J. Almeida, and A.Z. Broder. Summary cache: a scalable wide area web cache sharing protocol. *Transactions on networking*, 8(3), 2000.
- [9] R. Friedman. Caching web services in mobile ad-hoc networks. *Proc. of the 2nd Int. Workshop of Principals of Mobile Computing (POMC'02)*, pages 90–96, 2002.
- [10] Gnutella. Gnutella website. <http://gnutella.wego.com>.
- [11] T. Herman and S. Tixeuil. A distributed TDMA slot assignment for wireless sensor networks. *Technical Report no.1370, LRI, Universit Paris SUD*, 2003.
- [12] C. Ho, K. Obraczka, G. Tsudik, and K. Viswanath. Flooding for reliable multicast in multi-hop adhoc networks. *Proc. of the 3th Int. Workshop on Discrete Algorithms and Methods for MOBILE Computing and Communications (DialM'99)*, pages 64–71, 1999.
- [13] D.B. Johnson and D.A. Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.
- [14] S. Keshav. *An Engineering Approach to Computer Networking*. Addison Wesley, April 1997.
- [15] M.V. Marathe, H. Breu, H.B. Hund III, S.S. Ravi, and D.J. Rosenkrantz. Simple heuristics for unit disk graphs. *Networks*, 25:59–68, 1995.
- [16] J.M. Menaud, V. Issarny, and M. Banatre. A scalable and efficient cooperative system for web caches. *IEEE Concurrency*, 8(3), 2000.
- [17] S. Ni, Y. Tseng, Y. Chen, and J. Sheu. The broadcast storm problem in a mobile adhoc network. (*MOBICOM'99*), pages 151–162, 1999.
- [18] W. Peng and X. Lu. On the reduction of broadcast redundancy in mobile adhoc networks. *Proc. of the 1st ACM Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC'00)*, 2000.
- [19] W. Peng and X. Lu. An efficient broadcast protocol for mobile adhoc networks. *Journal of Science and Technology*, 2002.
- [20] C. Perkins. Ad hoc on demand distance vector (AODV) routing. *Internet Draft, draft-ietf-manet-aodv-00.txt, citeseer.nj.nec.com/article/perkins99ad.html*, 1997.
- [21] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large scale peer-to-peer systems. *Proc. of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, 2001.
- [22] R. RoyChoudhury, S. Bandyopadhyay, and K. Paul. A distributed mechanism for topology discovery in adhoc wireless networks using mobile agents. *Proc. of the 1st ACM*

- Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC'00)*, 2000.
- [23] F. Sailhan and V. Issarny. Cooperative caching in adhoc networks. *Mobile data management*, pages 13–28, 2003.
- [24] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIG/COMM*, pages 149–160, 2001.
- [25] Y. Tseng, S. Ni, and E. Shih. Adaptive approaches to relieving broadcast storms in a wireless multihop mobile ad-hoc network. *IEEE Transactions on Computers*, 52(5):545–557, 2003.
- [26] V. Valloppillil and K.W. Ross. Cache array routing protocol, internet draft. <http://ircache.nlanr.net/Cache/ICP/carp.txt>, 1998.
- [27] B. Williams and T. Camp. Comparison of broadcasting techniques for mobile adhoc networks. *Proc. of the 3rd ACM Symposium on Mobile Ad Hoc Networking and Computing (MOBIHOC'02)*, 2002.
- [28] J. Wu, M. Gao, and Stojmenovic. On calculating power-aware connected dominating sets for efficient routing in ad hoc wireless networks. *Proc. of the 30th International Conference on Parallel Processing (ICPP'01)*, pages 346–353, 2001.
- [29] J. Wu and H. Li. On calculating connected dominating set for efficient routing in ad hoc wireless networks. *Proc. of the 3th Int. Workshop on Discrete Algorithms and Methods for MOBILE Computing and Communications (DialM'99)*, pages 7–14, 1999.
- [30] P. S. Yu and E. A. MacNair. Performance study of a collaborative method for hierarchical caching in proxy servers. In *Proceedings of the 7th International WWW Conference*, April 1998.
- [31] Haas Z. A new routing protocol for the reconfigurable wireless networks. *ICUP'97*, 1997.
- [32] B. Zhao, J Kubiatiowicz, and A Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. *Technical REport UCB/CSD-01-1141, Computer Science U.C. Berkeley*, 2001.
- [33] Y. Zhu and T.Y. Cheung. A new distributed breadth-first search algorithm. *Information Processing Letters*, 25(5):329–334, July 1987.