MLS: An Efficient Location Service for Mobile Ad Hoc Networks

Roland Flury, Roger Wattenhofer
Computer Engineering and Networks Lab, ETH Zurich
CH-8092 Zurich, Switzerland
rflury@tik.ee.ethz.ch, wattenhofer@tik.ee.ethz.ch

ABSTRACT

MLS is a distributed location service to track the position of mobile nodes and to route messages between any two nodes. The lookup of nodes is achieved by searching in a hierarchy of pointers that each node maintains. We show that MLS has constant stretch for lookup requests. In contrast to previous work, we consider a concurrent setup where nodes are truly mobile and move even while messages are being routed towards them. We prove correctness and efficiency of MLS and determine the maximum speed at which the nodes might move, which is up to 1/15 of the routing speed. To the best of our knowledge, this is the first work that bounds the node speed, a necessity to prove the success of a lookup algorithm. We verified our theoretical results through extensive simulation and show that the average lookup stretch is around 6.

Categories and Subject Descriptors: C.2.2 [Computer-Communication Networks] Network Protocols; F.2.2 [Analysis of Algorithms and Problem Complexity] Nonnumerical Algorithms and Problems

General Terms: Algorithms, Performance

Keywords: Location Service, MANET, Mobility, Ad Hoc Geo–Routing

1. INTRODUCTION

Ad hoc networking is used to communicate between hosts in the absence of dedicated routing infrastructure, when messages are forwarded by intermediate hosts if the sender and receiver are out of communication range. The quality of such a routing algorithm can be measured by its stretch; that is, the length of the chosen route divided by the length of the optimal route should be as small as possible. In this paper we study ad hoc routing on a network of truly mobile nodes and introduce a routing algorithm with constant stretch.

For systems where each node is equipped with a location sensing device, geographic routing has received much attention recently and is considered to be the most efficient and scalable routing paradigm. In the simplest form of georouting, every node greedily forwards messages towards the neighbor closest to the destination node. However, these geographic routing algorithms assume that the sender knows the position of the destination node. This introduces a high storage overhead if each node keeps track of the position of all other nodes. Even more challenging is the situation with mobile nodes: In a mobile ad hoc network (MANET), nodes might be moving continuously and their location can change even while messages are being routed towards them. Clearly, a node cannot continuously broadcast its position to all other nodes while moving. This would cause an excessive message overhead.

In the home-based approach, each node is assigned a globally known home where it stores its current position. A sender first queries the home of the destination node to obtain the current position and then sends the message. This can be implemented using distributed or geographic hashing [15], and is a building block of many previous ad hoc routing algorithms, including [7, 14, 17, 18]. Despite of its broad usage, the home-based approach is not desirable, as it does not guarantee low stretch: The destination might be arbitrarily close to the sender, but the sender first needs to learn this by querying the destination's home, which might be far away. Similarly, a large overhead is introduced by moving hosts, which need to periodically update their homes, which might be far away. Even more important is the observation that the destination node might have moved to a different location by the time the message arrives. Thus, simultaneous routing and node movement require special consideration.

1.1 Our Contribution

In this work, we consider an arbitrarily formed deployment area populated by mobile nodes. This area might contain holes where no nodes are located. In the sequel, we allegorize this situation as a world where land masses represent the deployment areas of the nodes, and lakes denote holes. The mobile nodes might be moving continuously on the land areas, even while messages are being routed towards them.

We present a routing framework called MLS in which each node can send messages to any other node without knowing the position of the destination node. The routing (lookup) algorithm works hand in hand with a publish algorithm, through which moving nodes publish their current location on a hierarchical data structure. In this paper we formally prove that the stretch of the lookup algorithm is O(1) and show in extensive simulation that the constant hidden in the O()-notation is approximately 6. The amortized message cost induced by the publish algorithm of a moving node is $O(d \log d)$, where d is the distance the node moved. Again, our simulations show that the hidden constants of the O()-notation for the publish overhead are small, the average being $4.3 \cdot d \log d$. Finally, MLS only requires a small amount of storage on each node. For evenly distributed nodes, the storage overhead is logarithmic in the number of nodes (with high probability).

This paper formally proves the correctness of MLS for concurrent lookup requests and node movement. That is, while a message is routed, the destination node might move considerably, but the lookup stretch remains O(1). To prove this property, we derive the maximum node speed v_{max}^{node} at which nodes might move. We express this speed as a fraction of v_{min}^{msg}, the speed at which messages are routed. Clearly, if $v_{max}^{node} \geq v_{min}^{msg}$, a message may not reach its destination at all. As a main result of this paper, we show that MLS is correct if $v_{max}^{node} \leq v_{min}^{msg}/15$ in the absence of lakes[1]. I.e. we show that the lookup stretch remains O(1) even though the destination node might move at a speed up to 1/15 of the message speed. To the best of our knowledge, this is the first work that determines the maximum node speed to allow concurrent lookup and node movement.

1.2 Related Work

Routing on ad hoc networks has been in the focus of research for the last decade. The proposed protocols can be classified as proactive, reactive, or hybrid. Proactive protocols distribute routing information ahead of time to enable immediate forwarding of messages, whereas the reactive routing protocols discover the necessary information on demand. In between are hybrid routing protocols that combine the two techniques.

Much work has been conducted in the field of geographic routing where the sender knows the position of the destination. Face routing is the most prominent approach for this problem [5]. AFR [10] was the first algorithm that guarantees delivery in $O(d^2)$ in the worst case, and was improved to an average case efficient but still asymtotically worst case optimal routing in GOAFR$^+$ [9]. Similar techniques were chosen for the Terminode routing [4], Geo-LANMAR routing [6], and in [8]. All of them combine greedy routing with ingenious techniques to surround routing voids. Georouting is not only used to deliver a message to a single receiver, but also for geocasting, where a message is sent to all receivers in a given area. All these georouting protocols have in common that the sender needs to know the position of the receiver.

If we consider a MANET, a sender node needs some means to learn the *current* position of the destination node. A proactive location dissemination approach was proposed in DREAM [3], where each node maintains a routing table containing the position of *all* other nodes in the network. Each node periodically broadcasts its position, where nearby nodes are updated more frequently than distant nodes. In addition to the huge storage and dissemination overhead, DREAM does not guarantee delivery and relies on a recovery algorithm, e.g. flooding.

An alternative to the fully proactive DREAM is the hybrid home-based lookup approach, as utilized in [7, 14, 17, 18]. However, this approach does not allow for low stretch routing, as outlined in the introduction.

[1]In the presence of lakes, v_{max}^{node} is reduced by a factor equal to the largest routing stretch caused by the lakes.

Awerbuch and Peleg [2] proposed to use regional matchings to build a hierarchical directory server, which resembles our approach. However, to handle concurrent lookup and mobility, a *Clean Move Requirement* was introduced, which hinders nodes to move too far while messages are routed towards them. With other words, a lookup request can (temporarily) stop its destination node from moving. Furthermore, the lookup cost of [2] is polylogarithmic in the size of the network, which restrains scalability.

A novel position dissemination strategy was proposed by Li et al. in [12]: For each node n, GLS stores pointers towards n in regions of exponentially increasing size around n. In each of these regions, one node is designated to store n's position based on its ID. The lookup path taken by GLS is bounded by the smallest square that surrounds the sender and destination node. As outlined in [1], GLS cannot lower bound the lookup stretch and lacks support for efficient position publishing due to node movement. Xie et al. presented an enhanced GLS protocol called DLM in [19], and Yu et al. proposed HIGH-GRADE [20], which is similar to [13]. In contrast to GLS, in DLM and HIGH-GRADE most location pointers do not store the exact position of the corresponding nodes, which reduces the publish cost. Nevertheless, neither of them can lower bound the publish cost (e.g. due to the problem described in Figure 3) and they do not tackle the concurrency issue described above.

Recently, Abraham et al. proposed LLS [1], a locality aware lookup system with worst case lookup cost of $O(d^2)$, where d is the length of the shortest route between the sender and receiver. Similar to GLS, LLS publishes position information on a hierarchy of regions (squares) around each node. A lookup requests circles around the sender node with increasing radius until it meets one of the position pointers of the destination node, and then follows this pointer. MLS borrows some ideas from LLS and HIGH-GRADE, adding support for concurrent mobility and routing, improving the lookup to have linear stretch and bounding the publish overhead.

1.3 Outline

In the following sections, we present the MLS algorithm. We start with our model assumptions and the hierarchical lookup system through which messages are routed. Then, we discuss in more detail the routing of messages, and define a policy when a moving node needs to update its position data in Section 5. We introduce the issues of concurrent message routing and node movement in Section 6 and present the MLS algorithm in a concise and formal way in Section 7, such that we can proof its correctness in the sequel. Finally, we describe our simulation setup and conclude our work.

2. MODEL

For the analysis of our algorithm, we consider a world built of land and lakes. The nodes are distributed on the land areas, whereas no nodes can be placed on lakes. In order to allow for total connectivity, we assume that there are no islands, i.e. there are no disconnected land areas. The nodes are expected to contain a positioning system such as GPS, Cricket [16], cell tower or WLAN triangulation. Furthermore, each node is equipped with a communication module that provides reliable inter-node communication with minimal range r_{min}.

In our model, the nodes actively participate in ad hoc routing to deliver messages. To guarantee the reachability

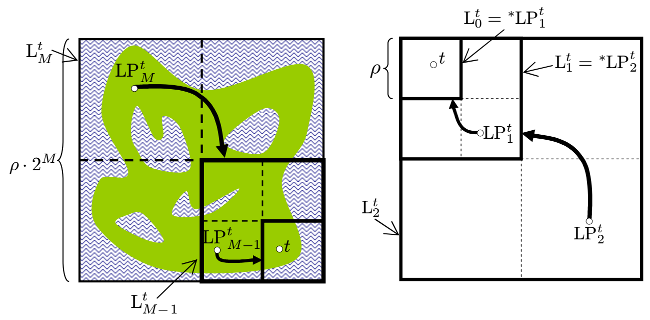

Figure 1: The left figure shows the entire world surrounded by L_M, the square of side length $\rho \cdot 2^M$. The land masses are filled with solid color (green), lakes are filled with waves. For each node t, L_i^t contains a level pointer LP_i^t that points to the sub square L_{i-1}^t of size $\rho \cdot 2^{i-1}$ that contains t. The right picture shows the smallest three levels around t and how each level for $i > 0$ contains a pointer that points to the next smaller level.

of any location on land, we consider a relatively dense node distribution and require that for any position p on land, there is a node at most $\lambda = r_{min}/3$ away. Furthermore, this invariant should hold over time while nodes are moving.

Using MLS, each node can send messages to any other node without knowing the position of the destination. To perform this task, MLS stores information about the nodes' whereabout in well defined positions (see Section 3). Then, the messages are routed to some of these special positions, where they learn about the current location of their destination. Because the positions of the intermediate destinations are known, this underlying routing might be performed by a geographic routing algorithm.

For the rest of this paper, we abstract from this low level routing and assume the following communication capability: For any two positions p_s and p_t on land, a node s in the λ-proximity of p_s can send a message to a node in the λ-proximity of p_t. The time to route the message is bounded by $\eta \cdot |p_s p_t|$, where $|p_s p_t|$ is the Euclidean distance[2] between p_s and p_t. We assume that this underlying routing algorithm selects the shortest path if it has a choice.

Note that this underlying routing capability is orthogonal to the main routing problem discussed in this paper, where the sender does not know the position of the receiver.

3. POSITION INFORMATION

In this section, we describe how each node t maintains a lookup system through which MLS routes messages to t. This lookup system is based on several layers, where the top layer is the smallest square of side length $\rho \cdot 2^M$ that encloses the entire world. M is dependent on the size of the world, and $\rho = \lambda/\sqrt{2} = r_{min}/(3\sqrt{2})$ is given by the radio range (see Section 2). In the following, we denote this square by level-M and write L_M. (Please refer to Table 1 for a summary of the notation used in this paper.) Similar to a geographic hash table (GHT) [15], each node has a designated position on land where it stores directions to its

L_i^t	Level that contains t with side-length $\rho \cdot 2^i$
$(L_i^s)^8$	The 8 surrounding squares of L_i^s
LP_i^t	Level pointer on L_i for node t; points to L_{i-1}
${}^*LP_i^t$	The L_{i-1} where LP_i^t points to
δ_i^t	distance of a node t to ${}^*LP_{i+1}^t$
FP_i^t	Forwarding pointer if $LP_i^t \notin {}^*LP_{i+1}^t$
${}^*FP_i^t$	The L_i where ${}^*FP_i^t$ points
TFP_i^t	Temporary forwarding pointer, before a pointer to t is removed
${}^*TFP_i^t$	The L_i where TFP_i^t points
TTL_i	Time to live of a TFP_i
v_{max}^{node}	Max. speed of nodes
r_{min}	Min. communication range of a node
λ	Min. distance to a node from any land point
ρ	Side length of L_0; $\rho = \lambda/\sqrt{2} = r_{min}/(3\sqrt{2})$
M	L_M surrounds the entire world
α	When $\delta_i^t \geq \alpha \cdot \rho \cdot 2^i$, LP_{i+1}^t is updated
$\beta(\beta_T)$	Max. number of forwarding hops to reach LP_i^t from a FP_i^t (TFP_i^t)
γ	See Lemma 8.2
η	Routing overhead to route to a given position

Table 1: Nomenclature used throughout the paper.

position. But instead of storing its exact position, a node t only stores in which of the four possible sub squares it is located, as depicted in Figure 1. Recursively, each selected square contains a pointer to its sub square that surrounds t. Finally, the chain is broken when the size of the sub square reaches ρ. Thus, a message for node t can be routed along these pointers until it reaches the smallest square.

We use the term L_{M-1} to denote any of the squares received when L_M is divided into 4 sub squares of side length $\rho \cdot 2^{M-1}$. Recursively, L_i denotes any square of side-length $\rho \cdot 2^i$ that can be obtained by dividing a L_{i+1} square into its 4 sub squares. The recursion stops for L_0, which is a square of side length ρ. To denote the L_i that surrounds a specific node t, we use the notation L_i^t. Clearly, L_M^x is the same for all nodes x, namely the square that surrounds the entire world.

On each L_i^t for $i > 0$, node t has a well defined position where it stores in which of the 4 possible L_{i-1} it is located. We call this information **L**evel **P**ointer and write LP_i^t to denote the level pointer on L_i^t that points to L_{i-1}^t. Also, we write ${}^*LP_i^t$ to denote the L_{i-1}^t where LP_i^t points.

We have seen that every node t stores a LP_i^t on each of its levels L_i^t. This information needs to be hosted on a node somewhere in L_i^t. But because the nodes are mobile, we cannot designate a specific node on each L_i^t to store the LP_i^t. Therefore, we propose to store this pointer at a specific position p on land in L_i^t. Due to the minimal node density, we know that there exists at least one node in the λ-proximity of p, which we can use to store LP_i^t. If there are several nodes in the λ-proximity of p, we pick the one closest to p. The selected node then hosts LP_i^t until it moves away from p by more than λ. At that point, it passes on LP_i^t to its neighbor node closest to p, which must exist due to the minimal node density. Over time, the LP_i^t is not necessarily stored on the node closest to p, but by an arbitrary node in the λ-proximity of p.

Each node t stores a LP_i^t at a well defined position p_t on land within L_i^t, where p_t is determined through the unique ID_t of t. Any consistent hash function that maps the ID of a node onto a position on land can be used for this purpose,

[2]Note that if $|p_s p_t| \rightarrow 0$, the message would have to be delivered instantaneously. However, unless the sender and receiver are identical, at least one hop is necessary, which requires time. We can safely ignore this boarder case because this paper presents a worst case analysis, where the shortest distance to be routed is λ.

as long as the chosen positions are evenly distributed over the land area for different IDs.

One possible function is the following, where we use two hash-functions $H_1()$ and $H_2()$ to map the ID of t to real numbers in the range $]0, 1]$. p_t is determined as an offset $(\Delta x, \Delta y)$ from the top left corner of L_i^t. Δy is chosen such that, when only considering L_i^t, the fraction of land above Δy is $H_1(ID_t)$. Once Δy is fixed, we must choose Δx such that p_t lies on land. We concatenate the line-segments where p_t can be placed to a single line and determine Δx such that the length of the line left to p_t is $H_2(ID_t)$ of the total line length.

Given this mapping, any node can determine the potential position p_t where a node t might store a LP_i^t for any L_i^t. All the node needs to know is the ID of the receiver and the position of the lakes. Amongst others, this is necessary to route a message along the level pointers towards t: Once a message has been routed to a LP_i^t, the node hosting LP_i^t determines p'_t in $^*LP_i^t$ and forwards the message to LP_{i-1}^t, which is located at p'_t.

We can already see that the number of levels only depends on the size of the deployment area and the transmission radius r_{min}. Therefore, every node needs to maintain only a constant number of level pointers. Because the positions of the level pointers are chosen randomly on the different levels, the storage overhead is balanced smoothly on the nodes if the nodes themselves are evenly distributed. This is an important property of `MLS`, and avoids overloading a few nodes with excessive amounts of data.

4. LOOKUP

Because messages are routed to a priori unknown positions, we denote them as *lookup requests*. When a sender node s wants to send a message to a destination node t, it issues a lookup request for node t, which encapsulates the message to be sent. So far, we have described where each node publishes its level pointers and how a lookup request is forwarded along the level pointers towards t once a first LP_i^t has been found. This section is devoted to the first phase of the lookup algorithm, which routes the lookup request to the first LP_i^t.

We propose a lookup algorithm that first searches t in the immediate neighborhood of s and then incrementally increases the search area until a LP_i^t is found. From there, the lookup request can be routed towards the smaller levels, as described in the previous section. Using this approach, we find t quickly if it is close to s. In particular, we prove that the lookup time is linear in the distance between s and t. As for the search areas, we use an extended version of the levels of node s, who issued the lookup request. For each L_i^s, we define $(L_i^s)^8$ to be the 8 L_i squares adjacent to L_i^s.

In the very first step of a lookup, node s checks whether t is in its immediate neighborhood. In this case, the message can be sent directly to t. Otherwise, the lookup request is sent to L_1^s to check whether it finds a LP_1^t. If this is not the case, the lookup request is forwarded in sequence to the 8 squares of $(L_1^s)^8$, where it tries to find LP_1^t. This step is repeated recursively: while the lookup request fails to find a LP_i^t on L_i^s and $(L_i^s)^8$, it is forwarded to L_{i+1}^s and then in sequence to squares of $(L_{i+1}^s)^8$, where it tries to find LP_{i+1}^t. A possible lookup path through the first 3 levels is depicted in Figure 2. Note that when the lookup request is forwarded through the levels of $(L_i^s)^8$, the sequence is chosen such that the last visited L_i is contained in L_{i+1}^s, and the request is

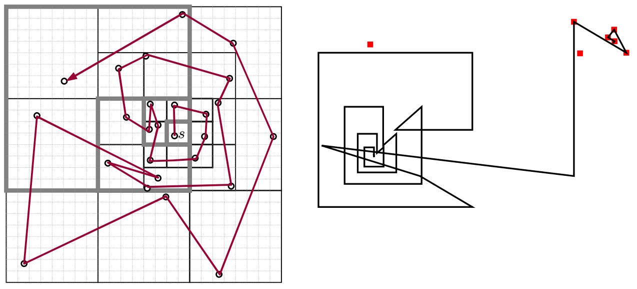

Figure 2: When s issues a lookup request for node t, the request is forwarded to the potential positions of a LP_i^t in $L_i^s \cup (L_i^s)^8$ for increasing i. The bold gray squares in the left image indicate the first 4 levels of node s. Note that we have only drawn the nodes visited by the chosen route. The right image shows a lookup path found by our simulation framework. The square dots indicate the LP of the destination node. Because no lakes were present in the lookup area, the lookup path is regular and draws quadratic shapes.

always forwarded to a neighboring L_i that shares an edge with the current L_i. (Skip L_i that are completely covered by lakes.)

In the first phase of the lookup algorithm, the lookup request is routed to a series of levels L_i, where it tries to find a level pointer LP_i^t. In the following lemma, we provide an upper bound for the time needed to search i levels. Note that this also gives an upper bound for the time needed to find LP_i^t on L_i, given that $LP_i^t \in L_i$.

LEMMA 4.1. *The accumulated time for searching a level pointer of node t on the levels 0 through i is bounded by $\eta 2^{i+1}\rho(\sqrt{2} + 8\sqrt{5})$.*

PROOF. When a lookup request for node t issued by a node s starts its search on L_j, it first queries for LP_j^t in L_j^s. From the lookup algorithm presented above, we know that the lookup request tries[3] to end its search of the levels L_{j-1} on a node in L_j^s. Thus, in the worst case, the request has to be routed over the diagonal of L_j^s to reach the potential place of LP_j^t, which implies a maximal route-time of $\eta 2^j \rho\sqrt{2}$. Then, to check for LP_j^t in $(L_j^s)^8$, the lookup request is repeatedly sent to a neighboring L_i to which the previous L_i shares an edge. At the worst, this takes $\eta 2^j \rho\sqrt{5}$ for each of the 8 neighbors. Thus, the total time to query for LP_j^t on level j is at most $\eta 2^j \rho(\sqrt{2}+8\sqrt{5})$, and the accumulated time to query i levels is bounded by $t \leq \sum_{j=0}^{i} \eta 2^j \rho(\sqrt{2} + 8\sqrt{5}) < \eta 2^{i+1}\rho(\sqrt{2} + 8\sqrt{5})$. □

Because L_M is the same square for all nodes, we are sure that a lookup request finds a level pointer for t at the latest on L_M^s [4]. In a later section, we give an upper bound on the time the lookup request needs to find a first LP_i^t based on the distance between s and t.

[3]This fails, if L_{j-1}^s is the only sub-level of L_j^s covering land. In this case, the lookup request first needs to move into L_j^s. This additional overhead is well compensated on the previous level, where at least 3 L_{j-1}^s were not visited.

[4]This holds also under lazy publishing and in the concurrent setting, two concepts that are introduced in the following sections.

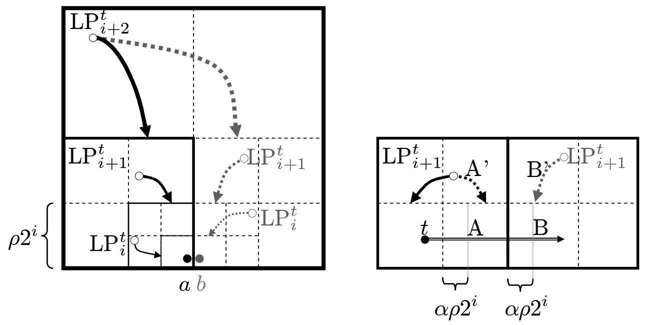

Figure 3: If a node t oscillates between the two points a and b in the left picture, immediate updating of its level pointers would cause an enormous amount of traffic. In black are the level pointers necessary if t is at a, the level pointers necessary at b are gray. Lazy publishing delays the updating of LP^t_{i+1} until node t has moved away from $^*\mathrm{LP}^t_{i+1}$ by more than $\alpha\rho 2^i$, as depicted in the right picture. Only when t moves across A, LP^t_{i+1} is updated to A'. Similarly, only when t crosses B, A' is deleted and B' is added.

5. LAZY PUBLISHING

In this and the following section, we analyze the implications of mobile nodes. In particular, this section focuses on the publish algorithm, through which every node keeps its level pointers up to date when it moves. Remember that every node t maintains a LP^t_i on each of its L^t_i for $0 < i \leq M$. Clearly, if a node t must update its LP^t_{i+1} as soon as it changes L^t_i, the publish cost might be extremely high. The left part of Figure 3 depicts a situation where many level pointers would have to be updated due to an arbitrarily small move of node t. If t oscillates between the two points a and b, and immediately sends messages to update the level pointers after moving an ϵ-distance, an enormous amount of traffic would be generated. To reduce this overhead, we employ lazy publishing, a concept that is similar to the lazy update technique utilized in [1].

Lazy publishing allows a node t to move out of $^*\mathrm{LP}^t_{i+1}$ up to a certain distance without updating LP^t_{i+1}, and reduces the overhead due to oscillating nodes. The following publish policy defines when a node t needs to update its LP^t_i.

We use the notation δ^t_i to denote the air distance of node t to $^*\mathrm{LP}^t_{i+1}$. Formally, δ^t_i is the shortest distance of t to any edge of $^*\mathrm{LP}^t_{i+1}$ if $t \notin {}^*\mathrm{LP}^t_{i+1}$. Otherwise, $\delta^t_i = 0$.

Definition 5.1. (Publish Policy) *When a node t has moved away from $^*\mathrm{LP}^t_{i+1}$ by more than $\alpha \cdot \rho \cdot 2^i$, it needs to update LP^t_{i+1}, such that LP^t_{i+1} points to the current L^t_i. Formally, a node t must update LP^t_{i+1} if $\delta^t_i \geq \alpha \cdot \rho \cdot 2^i$, for a fixed $\alpha \in \mathbb{R}^+$.*

We need to consider two cases while updating a LP^t_{i+1}: If the outdated LP^t_{i+1} is in L^t_{i+1}, it suffices to change the value of LP^t_{i+1} such that it points to L^t_i. In the right half of Figure 3, this is the case when t moves over the line marked A. However, if t has moved to a different L_{i+1}, the outdated LP^t_{i+1} needs to be removed and a new one must be added to L^t_{i+1}. An example of this case is depicted in Figure 3, when t moves over the line marked B.

The implementation of such an update is straight forward: Node t sends an *update* message to the outdated LP^t_{i+1} to change its value. If a new LP^t_{i+1} needs to be created, t sends a *remove* message to the outdated LP^t_{i+1} and a *create* message to the position of the new LP^t_{i+1}.

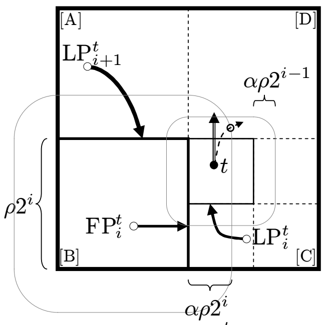

Figure 4: A node t is outside $^*\mathrm{LP}^t_{i+1}$, but does not need to update LP^t_{i+1}. Instead of removing its LP^t_i in $^*\mathrm{LP}^t_{i+1}$, t maintains a FP^t_i that points to the neighboring L_i there LP^t_i can be found. When t moves upwards along the indicated (solid) path, it eventually leaves $^*\mathrm{LP}^t_i$ by more that $\alpha\rho 2^{i-1}$ and needs to add a new LP^t_i in [D], update FP^t_i in [B] and remove the LP^t_i in [C]. To prevent a racing condition with a concurrent lookup, LP^t_i in [C] is not removed, but transformed to a TFP^t_i that points to [D].

6. CONCURRENCY

Clearly, the lookup algorithm presented in Section 4 does not support lazy publishing in its second phase, where the lookup request follows the level pointers in order to find a destination node t. So far, we have assumed that for $i > 1$, $^*\mathrm{LP}^t_i$ contains a LP^t_{i-1}. However, under the lazy publishing policy, LP^t_{i-1} might be outside $^*\mathrm{LP}^t_i$. We now derive modifications to the lookup and publish algorithms, such that they support lazy publishing. At the same time, we introduce the issue of simultaneous lookup and publish requests.

6.1 Forwarding Pointer

MLS uses a forwarding pointer to guide a lookup request to LP^t_i. If $\mathrm{LP}^t_i \notin {}^*\mathrm{LP}^t_{i+1}$, $^*\mathrm{LP}^t_{i+1}$ contains a forwarding pointer at the location where the LP^t_i would be. This forwarding pointer points to the neighboring L_i that contains LP^t_i. In the following, we denote such a forwarding pointer by FP^t_i and write $^*\mathrm{FP}^t_i$ to denote the L_i where FP^t_i points. Figure 4 depicts a situation where a FP is necessary. Note that we restrict the value of α to the range [0, 1[such that a FP^t_i might only point to an adjacent L_i. This simplifies our analysis, but does not restrain the final result, where α is chosen clearly smaller than 1 in order to maximize the allowed speed at which nodes move.

With the forwarding pointers, a lookup request has an easy means to find the LP^t_i if $\mathrm{LP}^t_i \notin {}^*\mathrm{LP}^t_i$. However, this simplistic approach only works in a static setup, where all publish requests of node t have terminated before a lookup request queries for t. Consider again Figure 4 and suppose that t moves upwards. When $\delta^t_i > \alpha\rho 2^{i-1}$, t needs to send three messages: one to remove LP^t_i from [C], one to add a new LP^t_i in [D] and one to change the direction of FP^t_i in [B]. Because the messages might be delayed randomly by the presence of lakes, it is possible that a lookup request reads FP^t_i before it is updated, and then fails to find LP^t_i in $^*\mathrm{FP}^t_i$ because LP^t_i was already removed. This is a racing condition between a lookup request and the publish request which we need to avoid.

6.2 Temporary Forwarding Pointer

Inspired by the fact that the racing condition in the previous example arose because of a LP^t_i that was removed too early, MLS does not remove LP_i^t immediately, but leaves behind a **T**emporary **F**orwarding **P**ointer, denoted TFP_i^t. Such a TFP_i^t points to the neighboring L_i where t was located when it decided to remove the LP_i^t. We will write $^*\mathrm{TFP}_i^t$ to denote the L_i where the TFP_i^t points.

To come back to Figure 4, t changes the LP_i^t in [C] to a TFP_i^t instead of removing it. A lookup request that follows the outdated FP_i^t then might find such a TFP_i^t instead of the expected LP_i^t and follows the TFP_i^t to finally find LP_i^t in [D].

At the time when LP_{i+1}^t is updated, the FP_i^t in $^*\mathrm{LP}_{i+1}^t$ becomes obsolete and needs to be removed. But deleting the FP_i^t could cause similar racing conditions with a concurrent lookup request as when a LP_i^t is removed. Therefore, MLS overwrites a FP_i^t with a TFP_i^t instead of removing the FP_i^t.

According to its name, a TFP does only exist for a limited time. TFP are automatically removed after a given time, which we denote TTL_i for a TFP_i^t. Thus, the lifetime TTL_i of a TFP_i^t depends on the value of i. We give constraints on the value of TTL_i while proving the correctness of MLS. Given that TTL_i can be determined statically, a TFP_i^t can be removed by the hosting node without any interaction of node t.

7. THE MLS ALGORITHM

We have now gathered all parts of MLS and present it here in a concise form before proving its correctness and performance. As before, the algorithm comes in two parts, the publish and the lookup algorithm. The publish algorithm is executed permanently by each moving node as to maintain valid information on its hierarchy of levels, whereas the lookup algorithm is used to route a message to a given node.

7.1 MLS Publish

During the startup phase of node t, initialize all level pointers LP_{i+1}^t to point to L_i^t.
While t is moving, it executes the following code:

```
1  if($\delta_i^t \geq \alpha \cdot \rho \cdot 2^i$) {
2      if(i>0) {change $\mathrm{FP}_i^t$ in $^*\mathrm{LP}_{i+1}^t$ to $\mathrm{TFP}_i^t$;}
3      if($\mathrm{LP}_{i+1}^t \in \mathrm{L}_{i+1}^t$) {
4          change $\mathrm{LP}_{i+1}^t$ to point to $\mathrm{L}_i^t$;
5      } else {
6          if($\mathrm{LP}_{i+1}^t \in {}^*\mathrm{LP}_{i+2}^t$) {
7              change $\mathrm{LP}_{i+1}^t$ to $\mathrm{FP}_{i+1}^t$ that points to $\mathrm{L}_{i+1}^t$;
8          } elseif($\mathrm{L}_{i+1}^t = {}^*\mathrm{LP}_{i+2}^t$) {
9              change $\mathrm{LP}_{i+1}^t$ to $\mathrm{TFP}_{i+1}^t$ that points to $\mathrm{L}_{i+1}^t$;
10         } else {
11             change $\mathrm{LP}_{i+1}^t$ to $\mathrm{TFP}_{i+1}^t$ that points to $\mathrm{L}_{i+1}^t$;
12             change $\mathrm{FP}_{i+1}^t$ to point to $\mathrm{L}_{i+1}^t$;
13         }
14         on $\mathrm{L}_{i+1}^t$, add $\mathrm{LP}_{i+1}^t$ that points to $\mathrm{L}_i^t$;
15     }
16     if(i>0 and $\mathrm{LP}_i^t \notin \mathrm{L}_i^t$) {
17         add $\mathrm{FP}_i^t$ on $\mathrm{L}_i^t$ that points to $\mathrm{L}_i \ni \mathrm{LP}_i^t$;
18     }
19 }
```

In the initialization phase, t sends a message to the position of LP_i^t on each L_i. This message tells the receiving node to store LP_i^t, which points to L_{i-1}^t. We assume that there are no lookup requests for t during this initial phase.

While t is moving, it utilizes lazy publishing (line 1). If t updates LP_{i+1}^t, FP_i^t becomes obsolete and is changed to a TFP_i^t that points to L_i^t (line 2). The simplest case arises when only the value of LP_{i+1}^t needs to be changed, because LP_{i+1}^t can still point to the current L_i^t (lines 3,4). This corresponds to the situation in the right part of Figure 3 where t crosses A. Otherwise, LP_{i+1}^t is added to the current L_{i+1} of t (line 14). In between, we update the forwarding pointers according to the three different cases: (**1**) If the old LP_{i+1}^t is on the lookup-path because it is in $^*\mathrm{LP}_{i+2}^t$, it is modified to a FP_{i+1}^t (lines 6,7). (**2**) If t returned to $^*\mathrm{LP}_{i+2}^t$, we replace the old LP_{i+1}^t with a temporary forwarding pointer to $^*\mathrm{LP}_{i+2}^t$ (lines 8,9). The FP_{i+1}^t in $^*\mathrm{LP}_{i+2}^t$ is implicitly overwritten by line 14. (**3**) In all other cases where t moves from one square to another in $(^*\mathrm{LP}_{i+2}^t)^8$, we replace the old LP_{i+1}^t with a temporary forwarding pointer to the new L_i and update FP_{i+1}^t, which is located in $^*\mathrm{LP}_{i+2}^t$ (lines 11,12). Finally, if the new LP_{i+1}^t does not point to the L_i that contains LP_i^t, a forwarding pointer is added to L_i^t. This FP_i^t points to the L_i that contains LP_i^t (lines 16, 17). This last case arises for example in Figure 4 when t moves along the dashed path, such that the publish algorithm is triggered at the circled area. In that case, a FP_i^t is added in square [D] and points to square [C], which holds LP_i^t.

When t needs to update a pointer (LP_i^t or FP_i^t) on an arbitrary L_i, t sends a command message to the node that hosts the pointer, where the command message indicates how the pointer should be modified. In order to create a new pointer on a L_i, t sends a create message to the node closest to the position p_t in L_i. If t sends a create message to set a LP_i^t or FP_i^t at position p_t, but there already exists a pointer (LP_i^t, FP_i^t or TFP_i^t) at this position, the existing pointer is overwritten.

7.2 MLS Lookup

A lookup request for node t issued by a node s is routed according to the following code:

```
1  if($t \in \mathrm{L}_0^s \cup (\mathrm{L}_0^s)^8$) { exit(); }
2  for(i=1; true; i++) {
3      if($P_i^t \in \mathrm{L}_i^s$ || $P_i^t \in (\mathrm{L}_i^s)^8$) {
4          $p = P_i^t$;
5          break;
6      }
7  }
8  Follow $p$ until $\mathrm{LP}_1^t$ is reached.
9  Route to a node closest to an arbitrary point on land in $^*\mathrm{LP}_1^t$.
10 Forward to $t$.
```

If the destination node t is in the same unit square or an immediate neighbor, node s and t can communicate directly over their radio. Because s needs to know all its neighbors in $\mathrm{L}_0^s \cup (\mathrm{L}_0^s)^8$ for routing, this situation can be detected immediately and the lookup stops (line 1).

Then, for increasing size of the levels, s searches a pointer of t in L_i^s and then in the squares of $(\mathrm{L}_i^s)^8$. Note that the lookup request accepts any kind of pointer of node t, whether it is an LP_i^t, FP_i^t or TFP_i^t. Furthermore, remember from Section 4 that the squares of $(\mathrm{L}_i^s)^8$ need to be accessed in a given order.

In the second phase of the lookup algorithm, the lookup request is routed along the pointers until it reaches LP_1^t (line 8). Because $^*\mathrm{LP}_1^t$ does not contain a LP_0^t, the lookup picks an arbitrary position p on land in $^*\mathrm{LP}_1^t$ and routes the lookup request to the node closest to p (line 9). From that node, the lookup can be sent directly to t (line 10).

8. ANALYSIS

We devote this section to the analysis of `MLS`. In particular, we show that `MLS` works in a concurrent setup, where publish requests and lookup requests occur simultaneously. We prove that a lookup request finds its destination in $O(d)$ hops, where d is the distance between the sender and the destination. Also, we show that the amortized cost for publishing the position data is $O(d \log d)$, where d denotes the distance a node has moved. In order to prove these properties, we need to limit the maximum speed of nodes, denoted v_{max}^{node}. Throughout the proofs, we introduce different constraints on the value of v_{max}^{node}. After proving the correctness of `MLS`, we determine the maximum node speed that satisfies all these constraints.

We base the lookup performance on the distance $|st|$ between two nodes s and t. In the static case, this distance is well defined during an entire lookup operation. However, in the concurrent setting, both nodes, s and t, might be moving while a lookup request is executing, and the distance $|st|$ changes over time. In our analysis of the lookup algorithm, we determine the distance $|st|$ when a lookup request is issued and base the performance analysis on this value.

8.1 Lookup Analysis

For the first phase of the lookup algorithm, we show that a lookup request for a node t can be routed such that it finds a level pointer to node t. Then, for the second phase of the lookup algorithm, we prove that the lookup request can be routed along the pointers of t to finally reach t.

Lookup – Phase 1

We consider a lookup request for a node t issued by a node s, where $d = |st|$ is the distance between s and t at the moment when s issues the request. In this section, we prove that the time needed to find a first location pointer for t is $O(d)$. To start, we give a lower bound on TTL_i such that a lookup request cannot miss a pointer[5] to t due to concurrency. Then, we show that the lookup request meets a pointer to t at the latest while visiting $\mathrm{L}_{k+1}^s \cup (\mathrm{L}_{k+1}^s)^8$ for a given k dependent on d.

LEMMA 8.1. *Given that t is (and remains) located in $\mathrm{L}_i^s \cup (\mathrm{L}_i^s)^8$ and maintains a LP_i^t in this area, a lookup request issued by s finds a pointer to t at the latest while visiting $\mathrm{L}_i^s \cup (\mathrm{L}_i^s)^8$, if $\mathrm{TTL}_i \geq \eta 2^i \rho(\sqrt{2} + 8\sqrt{5})$.*

PROOF. When t needs to relocate its LP_i^t, it sends a message m to create a new LP_i^t on L_i^t, which takes at most $\Delta t_m \leq \eta 2^i \rho \sqrt{2}$ time to arrive (traversing L_i^t). At the same time, t sends a message m' to transform the outdated LP_i^t to a FP_i^t or TFP_i^t. For this worst case analysis, we assume that m' is delivered instantaneously.[6]

A lookup request fails to find LP_i^t if it interleaves with these publish messages such that it arrives at the position p of the new LP_i^t before m, and if it reaches the position p' of the outdated LP_i^t after m'. In the worst case, t relocates LP_i^t into L_i^s at time T_0 and the lookup request visits p just before m arrives at $T_1 < T_0 + \Delta t_m$. The lookup request then continues its search in the squares of $(\mathrm{L}_i^s)^8$ and might choose the path such that it visits p' only on its last step, which is at the latest after $\Delta t_{lookup} \leq \eta 2^i \rho 8\sqrt{5}$ (see proof of Lemma 4.1). Thus, the lookup request reaches p' at the latest at $T_2 \leq T_1 + \Delta t_{lookup} < T_0 + \eta 2^i \rho(\sqrt{2} + 8\sqrt{5})$. Even if the outdated LP_i^t was transformed to a TFP_i^t at T_0, the TFP_i^t exists at least until $T_0 + \mathrm{TTL}_i > T_2$ and the lookup request finds the TFP_i^t. (Note that if the outdated LP_i^t is transformed to a FP_i^t, the time during which p' hosts a pointer to t is even longer, because the FP_i^t is transformed to a TFP_i^t after t updates LP_{i+1}^t.) $\square$

In the static case where publish requests and lookup requests do not interfere, a lookup request finds a pointer to t at the latest while visiting $\mathrm{L}_{k+1}^s \cup (\mathrm{L}_{k+1}^s)^8$, if the side length of L_k is at least d ($d \leq \rho 2^k$). (We can argue that $s \in \mathrm{L}_{k+1}^s$ and thus t is at most d away from L_{k+1}^s. At the same time, the distance from t to any L_k outside $\mathrm{L}_{k+1}^s \cup (\mathrm{L}_{k+1}^s)^8$ is at least $\rho 2^k \geq d$. Because $\alpha < 1$, t must have created a LP_{k+1}^t in $\mathrm{L}_{k+1}^s \cup (\mathrm{L}_{k+1}^s)^8$.)

For the concurrent case, we weaken this result and show that the lookup request finds a pointer to t at the latest while visiting $\mathrm{L}_{k+\gamma}^s \cup (\mathrm{L}_{k+\gamma}^s)^8$, where $\rho 2^k \geq d > \rho 2^{k-1}$ and $\gamma \in \mathbb{N}^+$, if the maximum node speed is bounded by

$$v_{max}^{node} < \frac{1 - \frac{\alpha}{2} - 2^{-\gamma}}{\eta\sqrt{2}} \qquad (1)$$

and the temporary forwarding pointers exist long enough:

$$\mathrm{TTL}_i \geq \eta 2^{i+1} \rho(1/\sqrt{2} + 8\sqrt{5}) \qquad (2)$$

Note that increasing the value of γ results in a higher node speed, but a lookup request might need to search longer until it finds a first pointer to t. We keep $\gamma \geq 1$ as a parameter of `MLS` to tune its performance.

LEMMA 8.2. *Consider a lookup request for t issued by s and let d be the distance $|st|$ at the moment when the request is issued. For any $\gamma \geq 1$, $k \in \mathbb{N}$ such that $\rho 2^{k-1} < d \leq \rho 2^k$, and if the Equations (1) and (2) hold, the lookup request finds a pointer to t at the latest on one of the levels in $\mathrm{L}_{k+\gamma}^s \cup (\mathrm{L}_{k+\gamma}^s)^8$.*

PROOF. In a first step, we show that t has created a $\mathrm{LP}_{k+\gamma}^t$ in $\mathrm{L}_{k+\gamma}^s \cup (\mathrm{L}_{k+\gamma}^s)^8$ at the latest when the lookup request is issued. This property is necessary to ensure that the lookup request cannot arrive too early on $\mathrm{L}_{k+\gamma}$ and miss $\mathrm{LP}_{k+\gamma}^t$ because it is not yet created.

By definition, $s \in \mathrm{L}_{k+\gamma}^s$ and thus t is at most $d \leq \rho 2^k$ away from $\mathrm{L}_{k+\gamma}^s$. At the same time, the distance from t to any $\mathrm{L}_{k+\gamma-1}$ outside $\mathrm{L}_{k+\gamma}^s \cup (\mathrm{L}_{k+\gamma}^s)^8$ is at least $\rho(2^{k+\gamma} - 2^k)$. Furthermore, we know that t sent off a message to create a $\mathrm{LP}_{k+\gamma}^t$ in $\mathrm{L}_{k+\gamma}^s \cup (\mathrm{L}_{k+\gamma}^s)^8$ when it entered this region by more than $\alpha 2^{k+\gamma-1}\rho$ (lazy publishing). Thus, t moved at least a distance $\Delta d = \rho 2^k(2^\gamma(1 - \frac{\alpha}{2}) - 1)$ after sending off the message to create $\mathrm{LP}_{k+\gamma}^t$ and when the lookup request was issued.

At the limit, the update message to create $\mathrm{LP}_{k+\gamma}^t$ needs to be sent across $\mathrm{L}_{k+\gamma}$ which needs $\Delta t_{update} \leq \eta 2^{k+\gamma}\rho\sqrt{2}$. But the lookup request is only issued after t has moved Δd, which takes at least $\Delta t_{move} \geq \frac{\Delta d}{v_{max}^{node}}$. By Equation (1), $\Delta t_{move} > \eta 2^{k+\gamma}\rho\sqrt{2} \geq \Delta t_{update}$, which shows[7] that the

[5]A pointer to t is either a LP^t, a FP^t or a TFP^t.

[6]There might also be a message to update a potential FP_i^t, which is of no importance for this proof.

[7]Note that we did not consider that the lookup request needs some time to visit the levels $1...k + \gamma - 1$, because there are situations where this time is negligible small. Including this time would allow for a slightly better v_{max}^{node}, but unnecessarily complicate the proofs.

lookup request is issued after a $\mathrm{LP}^t_{k+\gamma}$ has been created in $\mathrm{L}^s_{k+\gamma} \cup (\mathrm{L}^s_{k+\gamma})^8$.

To conclude the proof, we show that a $\mathrm{TFP}^t_{k+\gamma}$ lives long enough to catch all cases where the lookup request does not find a $\mathrm{LP}^t_{k+\gamma}$. From Lemma 8.1 we know that the lookup request finds a pointer to t while t remains inside $\mathrm{L}^s_{k+\gamma} \cup (\mathrm{L}^s_{k+\gamma})^8$. Before t can relocate $\mathrm{LP}^t_{k+\gamma}$ outside $\mathrm{L}^s_{k+\gamma} \cup (\mathrm{L}^s_{k+\gamma})^8$, it must move at least $\Delta d' = \rho 2^k(2^\gamma(1+\frac{\alpha}{2})-1)$ after the lookup was issued, which takes at least $\Delta t \geq \frac{\Delta d'}{v^{node}_{max}} > \eta 2^{k+\gamma}\rho\sqrt{2}$ by (1). Thus, t might create a $\mathrm{TFP}^t_{k+\gamma}$ in $\mathrm{L}^s_{k+\gamma} \cup (\mathrm{L}^s_{k+\gamma})^8$ at least Δt after the lookup request was issued. By Lemma 4.1, the lookup request finishes visiting the levels of $\mathrm{L}_{k+\gamma}$ at the latest after $\Delta t_{lookup} \leq \eta 2^{k+\gamma+1}\rho(\sqrt{2}+8\sqrt{5})$. Thus, to ensure that the $\mathrm{TFP}^t_{k+\gamma}$ has not expired, it must have lived for at least $\Delta t_{lookup} - \Delta t$, which holds by Equation (2). $\square$

The previous lemma states that a lookup request can find a pointer to t. We now show that the lookup request meets another pointer to t when it is routed along a forwarding pointer (or temporary forwarding pointer). However, we need to restrain the maximum node speed to

$$v^{node}_{max} < \frac{\alpha}{\eta \cdot 2(3\sqrt{2}+\alpha)} \tag{3}$$

to ensure that this holds in all situations.

Lemma 8.3. *A lookup request that finds a FP^t_i or a TFP^t_i also finds a pointer to t in ${}^*\mathrm{FP}^t_i$ or ${}^*\mathrm{TFP}^t_i$, respectively, if the maximum node speed satisfies Equation (3).*

Proof. We need to show that **[a]** a pointer p' for t has been written in ${}^*\mathrm{FP}^t_i$ (${}^*\mathrm{TFP}^t_i$) before the lookup request following FP^t_i (TFP^t_i) arrives, and that **[b]** p' cannot expire before the lookup request arrives, if p' has been transformed to a temporary forwarding pointer. Let us denote the found FP^t_i (TFP^t_i) by p, the L_i that contains p by A, and the L_i where p points by B. Throughout the proof, we refer to the lines of the publish algorithm presented in Section 7.1.

The found pointer p was created by one of the lines 2, 7, 9, 11, 12, or 17. If $p = \mathrm{TFP}^t_i$ was created by line 2, it points to the L^t_i where t was located when t sent the message m to change the FP^t_i to p. By the lines 16-18, we know that if $B = {}^*\mathrm{TFP}^t_i$ did not contain LP^t_i, t sent a message m' to create a FP^t_i in B. If $p = \mathrm{FP}^t_i$ was created by line 17, $B = {}^*\mathrm{FP}^t_i$ contains LP^t_i by definition. If p was created by line 7, 9, 11, or 12, t sent a message m' to create a LP^t_i in B at the same time while sending a message m to create p.

Because t sends the message m' to create the necessary pointer p' in B no later than m that creates p, the lookup request cannot arrive at the location of p' in B before m'. Otherwise, m' would have been sent over a sub-optimal route contradicting the triangle inequality. Therefore, condition **[a]** holds.

Consider the case where p is a TFP^t_i and T_0 is the time when t sent message m to crate p. Then, B (where p points) is the L_i where t was located at T_0, and B contains as p' either a LP^t_i or a FP^t_i (lines 14, 16-18). Because t is located in B at T_0, it needs to move at least $\alpha 2^{i-1}\rho$ away from B until p' is changed to a TFP^t_i. Also, at T_0, t is at most $\alpha 2^i \rho$ away from A (line 1), and therefore m arrives no later than $T_1 \leq T_0 + \eta 2^i \rho(\sqrt{2}+\alpha)$ to create $p = \mathrm{TFP}^t_i$. At the limit, the lookup request reads p just before it expires at time $T_2 = T_1 + \mathrm{TTL}_i$, and then moves to p' in B. Because the air-distance $p - p'$ is bounded by $2^i\rho\sqrt{8}$, the lookup request might arrive at p' no later than $T_3 \leq T_2 + \eta 2^i \rho\sqrt{8} \leq T_0 + \mathrm{TTL}_i + \eta 2^i \rho(3\sqrt{2}+\alpha)$. By this time, p' must not be expired, which requires that it was created after $T_4 > T_3 - \mathrm{TTL}_i$. Before T_4, t moved at most $\Delta d = (T_4 - T_0) \cdot v^{node}_{max}$. By Equation (3), $\Delta d < \rho 2^{i-1}\alpha$, which shows that t has changed p' to a TFP^t_i after T_4 and that p' cannot expire before the lookup request arrives.

For the second case where p is a FP^t_i, we use the fact that p is changed to a TFP^t_i if t moves away from A by more than $\alpha \cdot 2^i\rho$ (lines 1,2), and therefore an update message m'' from t to p sent at T_0 arrives at the latest at $T_1 \leq T_0 + \eta 2^i \rho(\sqrt{2}+\alpha)$. If t moves out of B and modifies p' to a TFP^t_i at T_0, it sends a message m'' to p (line 2 or 12). In the worst case, the lookup request reads p just before m'' arrives and visits p'. Because the air distance $p - p'$ is bounded by $2^i\rho\sqrt{8}$, the lookup request arrives at p' no later than $T_2 \leq T_1 + \eta 2^i \rho\sqrt{8} \leq T_0 + \eta 2^i\rho(\alpha + 3\sqrt{2})$. From Equation (2), we know that $\mathrm{TTL}_i \geq \eta 2^{i+1}\rho(1/\sqrt{2} + 8\sqrt{5})$. Because $\alpha < 1$, p' cannot expire before the lookup request arrives. $\square$

We have shown that a lookup request can find a pointer to t and that it can follow pointers to find new pointers. In the following, we give upper bounds on the time needed to find a LP^t_i after a lookup request has been routed to an arbitrary FP^t_i or TFP^t_i. We tackle this problem by limiting the maximum number of forwarding hops that the lookup request has to follow until it reaches LP^t_i. If the lookup request finds a FP^t_i, we show that it is routed to the corresponding LP^t_i in at most β forwarding hops, if the maximum node speed is bounded by

$$v^{node}_{max} \leq \frac{\alpha(\beta-2)}{2\eta(\sqrt{2}+\alpha+\beta\sqrt{8})} \tag{4}$$

and $\beta > 2$. If the lookup request finds a TFP^t_i, we show that it is routed to the corresponding LP^t_i in at most β_T forwarding hops, if the maximum node speed is bounded by

$$v^{node}_{max} \leq \frac{\alpha 2^{i-1}\rho(\beta_T - 2)}{\eta 2^i \rho(\sqrt{2}+\alpha+\beta_T\sqrt{8}) + \mathrm{TTL}_i} \tag{5}$$

and $\beta_T > 2$. The values of β and β_T become two additional tuning parameters of `MLS` which influence v^{node}_{max} and the time a lookup needs to find its destination.

We will use the following helper lemma, which says that a node t needs to move at least a certain distance between consecutive updates to its LP^t_i.

Lemma 8.4. *Between successive updates to LP^t_i, node t moves at least $\Delta d_{update} \geq \alpha 2^{i-1}\rho$.*

Proof. When a node t updates LP^t_i, the new LP^t_i points to L^t_i, the L_i that contains t. Due to the publish policy (Definition 5.1), t only needs to update LP^t_i after it has moved out of L^t_i by $\alpha 2^{i-1}\rho$ and thus t must move at least Δd_{update} before it needs to update LP^t_i. $\square$

Lemma 8.5. *Given a lookup request for a node t that has found a FP^t_i, and that v^{node}_{max} satisfies Equation (4), the lookup request can be routed to the corresponding LP^t_i in at most $\Delta t \leq \beta\eta 2^i \rho\sqrt{8}$ for a given $\beta > 2$.*

Proof. Node t maintains FP^t_i as long as $\delta^t_i < \alpha 2^i \rho$ (lines 1,2 of the publish algorithm in Section 7.1) and t does not return to ${}^*\mathrm{LP}^t_{i+1}$ (line 9). When t updates its LP^t_i, it also sends an update to FP_i^t, which takes at most $\Delta t_{update} \leq \eta 2^i \rho(\sqrt{2}+\alpha)$ time to arrive. When $\delta_i^t = \alpha 2^i \rho$, t updates LP_{i+1}^t and changes the FP_i^t to a TFP_i^t (line 2). Thus, when the lookup request reads FP_i^t, it reads a direction that is at most Δt_{update} outdated.

Consider the case where a node t moved from level X to Y and sent an update m to FP_i^t, such that FP_i^t points to Y instead of X. If a lookup request reads FP_i^t before m arrives, it first visits XY, where it finds a TFP_i^t that points to Y [8]. By Lemma 8.3, this TFP_i^t has not yet expired.

For each forwarding (FP_i^t or TFP_i^t), the lookup request has to move an air distance bounded by $2^i\rho\sqrt{8}$. Thus, the last lookup request relayed by X arrives in Y at most $\Delta t_{update} + 2\eta 2^i \rho\sqrt{8}$ after m was sent off. But during this time, t might have moved - and left behind yet other (possibly temporary) forwarding pointers.

In order to limit the routing time to $\beta\eta 2^i\rho\sqrt{8}$, the lookup request can follow at most β forwarding pointers until it reaches LP_i^t. Thus, the total time between sending m and when the lookup request reaches LP_i^t is $\Delta t_{tot} \leq \Delta t_{update} + \beta\eta 2^i\rho\sqrt{8} \leq \eta 2^i \rho(\sqrt{2}+\alpha+\beta\sqrt{8})$. During this time, t moves at most $\Delta d \leq \Delta t_{tot} \cdot v_{max}^{node}$. By Equation (4), $\Delta d \leq \rho(\beta - 2)\alpha \cdot 2^{i-1}$ and causes at most $\Delta d / \Delta d_{update} \leq \beta - 2$ additional forwarding pointers by Lemma 8.4. Including the two forwarding hops to visit X and Y, the lookup request has to follow maximally β pointers, which takes at most $\beta\eta 2^i\rho\sqrt{8}$.

If ${}^*\mathrm{FP}_i^t$ does not contain a TFP_i^t that points to Y, then t must have returned to X and overwritten the TFP_i^t with a LP_i^t, a FP_i^t or a more recent TFP_i^t. Following such a FP_i^t or TFP_i^t short-cuts the path to LP_i^t, and the lookup request finds LP_i^t even faster. $\square$

LEMMA 8.6. *Given a lookup request for a node t that has found a* TFP_i^t*, and that* v_{max}^{node} *satisfies Equation (5), the lookup request can be routed to the corresponding* LP_i^t *in at most* $\Delta t \leq \beta_T \eta 2^i \rho\sqrt{8}$ *for a given* $\beta_T > 2$.

PROOF. We distinguish if the found TFP_i^t was created due to **[a]** line 9 or 11 of the publish algorithm in Section 7.1 or **[b]** by line 2. For both cases, we consider the time T_0 when t sends a message m to create TFP_i^t. Also, if the lookup request is forwarded along a FP_i^t (TFP_i^t), it finds the next pointer to t at the latest after $\Delta t_{forward} \leq \eta 2^i\rho\sqrt{8}$, because a FP_i^t (TFP_i^t) points to a neighboring L_i. Therefore, it is sufficient to show that the lookup request reaches LP_i^t at the latest after β_T forwarding hops.

For case **[a]**, t changed its old LP_i^t to a TFP_i^t because it has moved away from the L_i that contains LP_i^t by more than $\alpha 2^{i-1}\rho$. Thus, message m needs $\Delta t_{update} \leq \eta 2^i \rho(\alpha/2 + \sqrt{2})$ until it reaches the old LP_i^t. From line 14 and Lemma 8.3, we know that the lookup request finds a pointer to t in ${}^*\mathrm{TFP}_i^t$. Also, a lookup request that reads the TFP_i^t just before it expires, arrives in ${}^*\mathrm{TFP}_i^t$ at the latest at $T_1 = T_0 + \Delta t_{update} + \mathrm{TTL}_i + \Delta t_{forward}$, where it might not find LP_i^t (created due to line 14), because t has already moved away. The lookup request must catch up with t and find LP_i^t at the latest at $T_2 = T_0 + \Delta t_{update} + \mathrm{TTL}_i + \Delta t$. By this time, t has moved up to $\Delta d = (T_2 - T_0)v_{max}^{node}$, which is bounded by $\Delta d \leq (\beta_T - 2)\alpha 2^{i-1}\rho$ using Equation (5). From Lemma 8.4, we deduce that t has caused at most $\lfloor \frac{\Delta d}{\Delta d_{update}} \rfloor \leq \beta_T - 2$ (possibly temporary) forwarding pointers due to its motion, and the lookup request needs to follow at most a total of β_T pointers until it reaches LP_i^t.

We follow a similar argumentation for the second case **[b]**, where t changed a FP_i^t to the found TFP_i^t. At T_0, $\delta_i^t = \alpha 2^i\rho$ and thus the message m needs $\Delta t_{update} \leq \eta 2^i\rho(\alpha + \sqrt{2})$ until it reaches the outdated FP_i^t. Case **[b]** is also different in that ${}^*\mathrm{TFP}_i^t$ might never contain a LP_i^t, but only a FP_i^t (lines 16-18). Therefore, the lookup request might have to follow 2 pointers until it reaches LP_i^t, even if t does not move at all after T_0. Again, the lookup request must catch up with t and find LP_i^t at the latest at $T_2 = T_0 + \Delta t_{update} + \mathrm{TTL}_i + \Delta t$. By this time, t has moved up to $\Delta d = (T_2 - T_0)v_{max}^{node}$, which is bounded by $\Delta d \leq (\beta_T - 2)\alpha 2^{i-1}\rho$ using Equation (5). From Lemma 8.4, we deduce that t has caused $\lceil \frac{\Delta d}{\Delta d_{update}} \rceil \leq \beta_T - 2$ (possibly temporary) forwarding pointers due to its motion. (Note that we needed to round up the number of forwarding pointers because t did not update LP_i^t at T_0.) Therefore, the lookup request needs to follow at most a total of β_T pointers until it reaches LP_i^t. $\square$

We are now ready to assemble the first pieces of the puzzle and show that a lookup request finds a first LP^t in bounded time.

LEMMA 8.7. *Given that* v_{max}^{node} *satisfies the Equations (1), (3), (4) and (5), and* TTL_i *satisfies Equation (2) for fixed values of* $0 < \alpha \leq 1$, $\gamma \geq 1$, $\beta > 2$, *and* $\beta_T > 2$, *then, a lookup request for node t issued by node s finds a level pointer* LP^t *in* $\mathrm{O}(d)$ *time, where d is the distance* $|st|$ *at the moment when the request is issued.*

PROOF. By combination of the Lemmas 8.2, 8.5, and 8.6: A first pointer p to t is found at the latest in one of the squares $\mathrm{L}_u^s \cup (\mathrm{L}_u^s)^8$, where $u = \lceil \log_2 \frac{d}{\rho} \rceil + \gamma$ (Lemma 8.2). The necessary time the lookup request needs to visit all these levels is bounded by $T_1 \leq \eta 2^{u+1}\rho(\sqrt{2} + 8\sqrt{5})$ (Lemma 4.1). If p is a FP_i^t, the lookup request reaches the corresponding LP_i^t in $T_2 \leq \beta\eta 2^i\rho\sqrt{8}$ (Lemma 8.5), and if p is a TFP_i^t, the lookup request reaches the corresponding LP_i^t in $T_3 \leq \beta_T\eta 2^i \rho\sqrt{8}$ (Lemma 8.6). For T_2 and T_3, $i \leq u$. The total time T to find a first LP_i^t is bounded by

$$
\begin{aligned}
T &\leq T_1 + \max(T_2, T_3) \\
&\leq \eta 2^{\lceil \log_2 (d/\rho) \rceil + \gamma + 1}\rho(\sqrt{2} + 8\sqrt{5} + \max(\beta, \beta_T)\sqrt{2}) \\
&\leq d \cdot \underbrace{\eta 2^{\gamma+2}(\sqrt{2} + 8\sqrt{5} + \max(\beta,\beta_T)\sqrt{2})}_{constant} \\
&\in \mathrm{O}(d)
\end{aligned}
$$

$\square$

Lookup – Phase 2

For the second phase of the lookup algorithm, we need to show that once a lookup request has found a first LP_i^t, it can follow the pointers and find the destination node t. We start with another helper lemma stating that if LP_i^t points to L_{i-1} and t is at most $\alpha 2^{i-1}\rho$ away from L_{i-1}, then L_{i-1} contains a LP_{i-1}^t or a FP_{i-1}^t.

LEMMA 8.8. *As long as* $\delta_i^t < \alpha 2^i\rho$, ${}^*\mathrm{LP}_{i+1}^t$ *contains a node that hosts either a* LP_i^t *or a* FP_i^t.

PROOF. By inspection of the publish algorithm presented in Section 7.1. The only places where pointers are transformed to TFP_i^t are on the lines 2, 9 and 11. On line 2, a FP_i^t is removed, because LP_{i+1}^t will no longer point to the

[8]It is possible that ${}^*\mathrm{FP}_i^t$ does not contain a TFP_i^t that points to Y. We discuss this case later on.

level that contains the FP_i^t. But this only happens when $\delta_i^t \geq \alpha\rho 2^i$ (line 1).

Line 9 or 11 is executed when $\delta_{i-1}^t \geq \alpha\rho 2^{i-1}$. However, if line 9 or 11 executes, we know from line 6 that the LP_j^t that is overwritten is not in $^*\mathrm{LP}_{j+1}^t$. We conclude that the pointer for t in $^*\mathrm{LP}_{i+1}^t$ is transformed to a TFP_i^t iff $\delta_i^t \geq \alpha\rho 2^i$. $\square$

The following lemma states that a lookup request that has reached a LP_i^t can be routed along LP_i^t and find LP_{i-1}^t.

LEMMA 8.9. *Under the condition that $i > 0$, and all constraints on v_{max}^{node} and TTL_i are satisfied, a lookup request can follow LP_{i+1}^t and find LP_i^t after $\Delta t \leq \eta 2^i \rho\sqrt{8}(1+\max(\beta,\beta_T))$.*

PROOF. First, we show that $^*\mathrm{LP}_{i+1}^t$ contains a pointer p to t when the lookup request arrives. Then, we apply Lemma 8.5 and Lemma 8.6 to bound the time to find LP_i^t.

While $\delta_i^t < \alpha\rho 2^i$, $^*\mathrm{LP}_{i+1}^t$ contains a LP_i^t or a FP_i^t (by Lemma 8.8). At T_0, when $\delta_i^t \geq \alpha\rho 2^i$, the FP_i^t in $^*\mathrm{LP}_{i+1}^t$ is changed to a TFP_i^t (line 2 of the publish algorithm in Section 7.1) and a message m is sent to change LP_{i+1}^t, where it arrives at $T_1 \leq T_0 + \eta 2^i \rho(\alpha + \sqrt{8})$. A lookup request that follows the outdated LP_{i+1}^t before m arrives reaches the TFP_i^t in $^*\mathrm{LP}_{i+1}^t$ at the latest at $T_2 = T_1 + \eta 2^i \rho\sqrt{8} \leq T_0 + \eta 2^i \rho(\alpha + 4\sqrt{2})$. By Equation (2), $\mathrm{TTL}_i > T_2 - T_0$, and thus the TFP_i^t does not expire before the lookup request arrives.

So far, we have shown that the lookup finds a pointer to t in $^*\mathrm{LP}_{i+1}^t$. Following LP_{i+1}^t to reach p takes at most $\eta 2^i \rho\sqrt{8}$. If p is a FP_i^t, the additional time to route to LP_i^t is bounded by $\beta\eta \cdot 2^i \rho\sqrt{8}$ (Lemma 8.5). If p is a TFP_i^t, the additional time to route to LP_i^t is bounded by $\beta_T \eta 2^i \rho\sqrt{8}$ (Lemma 8.6). Thus, the total time to LP_i^t is upper-bounded by $\Delta t \leq \eta 2^i \rho\sqrt{8}(1 + \max(\beta, \beta_T))$. $\square$

Using the previous lemma, we can show that a lookup request can be routed from a LP_i^t to LP_{i-1}^t until it reaches LP_1^t, from where it is forwarded to $^*\mathrm{LP}_1^t$. It remains to verify that the lookup request can be sent directly to t from within $^*\mathrm{LP}_1^t$, which we show under the constraint that

$$v_{max}^{node} < \frac{\sqrt{2} - \alpha}{\eta(\alpha + 4\sqrt{2})} \tag{6}$$

LEMMA 8.10. *If v_{max}^{node} satisfies (6), a lookup request that has found LP_1^t can be sent directly to t from within $^*\mathrm{LP}_1^t$.*

PROOF. When t moves out of $^*\mathrm{LP}_1^t$ by more than $\alpha\rho$, it sends an update message m to change LP_1^t at time T_0 (Definition 5.1). This message arrives at $T_1 \leq T_0 + \eta\rho(\alpha + \sqrt{8})$. A lookup request that reads LP_1^t just before m arrives, is forwarded to $^*\mathrm{LP}_1^t$. Because $^*\mathrm{LP}_1^t$ does not contain a LP_0^t, the lookup request is routed towards an arbitrary point on land in $^*\mathrm{LP}_1^t$, and might end up on a node u that is up to λ away from $^*\mathrm{LP}_1^t$ (lines 8-10 of the lookup algorithm in Section 7.2). At the limit, this forwarding to u requires to traverse L_1^t, and the lookup request arrives at u at $T_2 \leq T_1 + \eta\rho\sqrt{8}$. By this time, t has moved $\Delta d \leq (T_2 - T_0)v_{max}^{node} < \rho(\sqrt{2} - \alpha)$ and is at most $\Delta d + \alpha\rho \leq \lambda$ away from $^*\mathrm{LP}_1^t$. Both, u and t are at most λ away from $^*\mathrm{LP}_1^t$, and the diameter of $^*\mathrm{LP}_1^t$ is $\rho\sqrt{2} = \lambda$. Thus, the total distance between u and t is at most $3\lambda = r_{min}$, which shows that u can forward the lookup request directly to t. $\square$

THEOREM 8.11. (LOOKUP) *Given that v_{max}^{node} satisfies the Equations (1), (3), (4), (5), and (6), and that TTL_i satisfies the Equation (2) for fixed values of $0 < \alpha \leq 1$, $\gamma \geq 1$, $\beta > 2$, and $\beta_T > 2$, then, a lookup request for a node t issued from a node s takes $\mathrm{O}(d)$ time to reach t, where $d = |st|$ is the distance between s and t when the request is issued.*

PROOF. By Lemma 8.7, we know that the time to find a first level pointer is bounded by $T_1 \leq d \cdot \eta 2^{\gamma+2}(\sqrt{2} + 8\sqrt{5} + \max(\beta, \beta_T)\sqrt{2})$ Also, we know that this first level pointer is found at the latest on level $\lceil \log_2 \frac{d}{\rho} \rceil + \gamma$. From Lemma 8.9, we deduce that following the level pointers from level $\lceil \log_2 \frac{d}{\rho} \rceil + \gamma$ downto 1 takes

$$T_2 \leq \sum_{j=1}^{\lceil \log_2 d/\rho \rceil + \gamma} \eta 2^j \rho\sqrt{8}(1 + \max(\beta, \beta_T))$$

and the time to forward the lookup request from LP_1^t to t can be bounded using Lemma 8.10 to

$$T_3 \leq \eta\rho\sqrt{8}(1 + \max(\beta, \beta_T))$$

The total time until the lookup request reaches t is therefore bounded by

$$\begin{aligned} T_{lookup} &\leq T_1 + T_2 + T_3 \\ &\leq T_1 + \sum_{j=0}^{\lceil \log_2 d/\rho \rceil + \gamma} \eta 2^j \rho\sqrt{8}(1 + \max(\beta, \beta_T)) \\ &< T_1 + d\eta 2^{\gamma+2}\sqrt{8}(1 + \max(\beta, \beta_T)) \\ &\leq d \cdot \underbrace{\eta 2^{\gamma+2}(3\sqrt{2}(1 + \max(\beta, \beta_T)) + 8\sqrt{5})}_{constant} \in \mathrm{O}(d) \end{aligned}$$

$\square$

8.2 Maximum Node Speed

While proving the lemmas in the previous section, we formulated constraints on the value of v_{max}^{node} and TTL_i. In this section, we collect all of these constraints and present the maximum node speed for which `MLS` is proven to work. The value of TTL_i only needs to satisfy Equation (2). Because enlarging TTL_i implies a reduction of v_{max}^{node} by Equation (5), we choose TTL_i as small as possible:

$$\mathrm{TTL}_i = \eta 2^{i+1} \rho(1/\sqrt{2} + 8\sqrt{5}) \tag{7}$$

To determine the maximum v_{max}^{node}, we need to fix the parameters $0 < \alpha < 1$, $\beta > 2$, $\gamma \geq 1$ and $\beta_T > 1$. The value of η is given indirectly by the topology and the underlying routing algorithm. While maximizing v_{max}^{node}, we opt to minimize T_{lookup}, the worst case time of a lookup request, which depends on the same parameters. Thus, we need to determine α, β, γ and β_T, such as to maximize v_{max}^{node} and minimize T_{lookup} under the constraints that the Equations (1), (3), (4), (5) and (6) are satisfied.

For $\gamma \to \infty$; $\beta \to \infty$, $\beta_T \to \infty$, we receive that $v_{max}^{node} \approx \frac{0.0845}{\eta}$ for $\alpha \approx 0.863$. This is an unreachable upper bound, because the maximum cost of a lookup request would grow to infinity. Clearly, there is a tradeoff between maximizing v_{max}^{node} and minimizing T_{lookup}. The higher we choose the maximum node speed, the longer is the worst-case time of a lookup request. But while T_{lookup} increases exponentially with γ, the value of γ barely increases v_{max}^{node}. Similarly, β and β_T have an impact on v_{max}^{node} while they are below 20, higher values only increase T_{lookup}. Twiddling the parameters, we found $\gamma = 1$, $\beta = 5$, and $\beta_T = 19$ to optimize the tradeoff between v_{max}^{node} and T_{lookup}. Using these values, v_{max}^{node} becomes maximal for $\alpha = 0.8$. We can now state our lower bound on the maximum node speed for which MLS is proven to work.

Theorem 8.12. *For $\alpha = 0.8$, $\beta = 5$, $\beta_T = 19$ and $\gamma = 1$, the nodes might move at speed $v_{max}^{node} \leq \frac{1}{15 \cdot \eta}$ without breaking MLS.*

Proof. By plugging the parameters into the Equations (1), (3), (4), (5) and (6). $\square$

In the absence of lakes, $1/\eta$ can be interpreted as the minimum speed at which messages are routed by the underlying routing algorithm. Therefore, the above result shows that nodes might move at a speed that is only 15 times smaller than the routing speed, which is remarkably fast. If we consider real-world nodes such as the mica2 nodes from UC Berkeley, a data packet experiences around 50 ms delay while being forwarded by a node. Thus, a packet can be sent about 40 hops per second. If we assume that a message is forwarded around 10 meters per hop, the message speed reaches around 400 meters per second. In this setup and in the absence of lakes, the maximum node speed is bounded by $400/16 = 25$ meters per second (90 km/h, about 56 mph), which exceeds by far the node speed in typical network applications.

8.3 Publish Analysis

Last but not least, we need to consider the cost of the publish algorithm. While a node is moving, it continuously sends updates onto its different levels. This produces messages that need to be routed by the nodes of the system. Thus, bounding the message overhead of the publish algorithm is of critical importance to ensure that the overall routing cost induced by the moving nodes is reasonable. First, we derive the maximum cost of publishing to a level. Then, we determine the amortized message cost for publishing while a node is moving.

Throughout the paper, we made use of an underlying routing algorithm that can deliver a message to its destination in $\eta \cdot d$ time, when the air distance between the sender and destination is d. For the following, we assume that the number of routing hops needed to send a message over a distance d is proportional to the routing time and write $\tilde{\eta} \cdot d$ to denote the number of routing hops needed to route a message to a target that is d away.

Lemma 8.13. *The cost to publish on level-i is bounded by $\tilde{\eta} 2^{i+2} \rho(\sqrt{8} + \alpha)$ message-hops.*

Proof. By inspection of the publish algorithm in Section 7.1. We determine the message cost c of each individual line to pick the execution with maximum cost.

On line 2, FP_i^t is in the neighboring level-i, which t just left by more than $\alpha \cdot 2^i$, thus $c_2 \leq \tilde{\eta} 2^i \rho(\alpha + \sqrt{2})$. For the lines 4, 7, 9, and 11, the distance from t to LP_{i+1}^t is maximally $2^i(\alpha + \sqrt{8})$ and the message cost is $c_{4,7,9,11} \leq \tilde{\eta} 2^i \rho(\alpha + \sqrt{8})$. FP_{i+1}^t on line 12 is contained in $(\mathrm{L}_{i+1}^t)^8$, but only as long as $\delta_{i+1}^t < \alpha 2^{i+1}$. Thus, the cost to send a message to FP_{i+1}^t is bounded by $c_{12} \leq \tilde{\eta} 2^i \rho(2\alpha + \sqrt{8})$. For line 14, LP_{i+1}^t is contained in L_{i+1}^t, and therefore the cost is $c_{14} \leq \tilde{\eta} 2^i \rho \sqrt{8}$. Finally, for line 17, t reaches FP_i^t in $c_{17} \leq \tilde{\eta} 2^i \rho \sqrt{2}$ hops, as $\mathrm{FP}_i^t \in \mathrm{L}_i^t$. The execution with maximum cost visits the lines 2, 11, 12, 14 and 17. Summing up the corresponding costs results in the indicated number of hops. $\square$

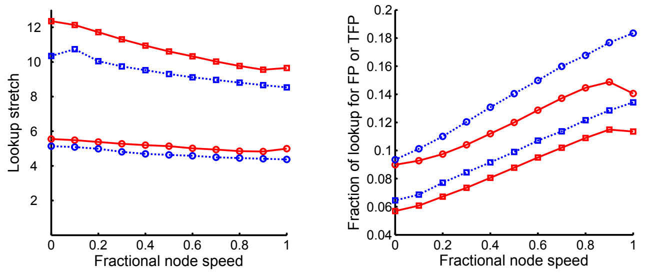

Figure 5: The left plot shows the lookup stretch depending on the node speed, where 1 is the maximum speed. The right plot shows the fraction of the time a lookup spends following FP and TFP. The solid lines indicate runs with Map$_1$ (a world as shown in Figure 1), the dotted lines show runs with Map$_2$ (no lakes). The results for the adapted fast-lookup are drawn with circles, the results for the sequential lookup with squares.

Using an amortized analysis, we can now show that the expected message overhead induced by the publish method can be bounded.

Theorem 8.14. (Publish) *The amortized message cost of a node induced by the publish method is at most $\mathrm{O}(d \log d)$ message hops, where d denotes the distance the node moved.*

Proof. A node t updates its LP_i^t when $\delta_{i-1}^t \geq \alpha\rho 2^{i-1}$ (Definition 5.1). After an update to LP_i^t, $t \in {}^*\mathrm{LP}_i^t$ and thus t needs to move at least $\alpha\rho 2^{i-1}$ before it needs to issue another update for LP_i^t (Lemma 8.4). Therefore, t needs to update LP_i^t at most $d/(\alpha\rho 2^{i-1})$ times while moving a distance d. The cost to publish on L_i is maximally $\tilde{\eta} 2^{i+2} \rho(\sqrt{8} + \alpha)$ (Lemma 8.13), and the total cost for updating L_i while moving the distance d is bounded by $d\tilde{\eta} 8\rho(\sqrt{8}+\alpha)/\alpha$. The total publish cost is given by the sum of the cost of each level to which t has to publish. Because $\forall j : \delta_j^t = 0$ after the initialization phase of the publish method, t only needs to publish on L_i if $d \geq \alpha\rho 2^{i-1}$, and the total publish cost is bounded by $d \log_2(2d/\alpha)\tilde{\eta} 8\rho(\sqrt{8} + \alpha)/\alpha \in \mathrm{O}(d \log d)$. Note that the total number of levels is M (Section 2) and the publish cost is further bounded by $dM\tilde{\eta} 8\rho(\sqrt{8} + \alpha)/\alpha$ $\square$

9. SIMULATION

We support our theoretical results with a series of simulations, through which we show that the average lookup time and publish cost are well below the worst case costs. Our simulation framework implements mobile nodes that select their route using the random waypoint model and maintain their hierarchical lookup data according to Section 7.1. However, being mainly a proof of concept, the simulation abstracts from the underlying routing[9], where we delay messages according to the route length, but do not simulate the node-to-node routing. Also, we simulate only a (random) sub-set of the nodes as to improve the run-time. As a consequence, the storage of pointers is performed by the framework and relieves us from ensuring the minimum node density.

The techniques of Le Boudec et al. [11] to obtain a stationary regime cannot be applied, because the nodes are not memoryless with respect to their lookup hierarchy. Therefore, we obtain a close to stationary regime through a long

[9]Remember that the underlying routing can route a message to a given *position*. This is orthogonal to our main goal, where the location of the destination node is unknown.

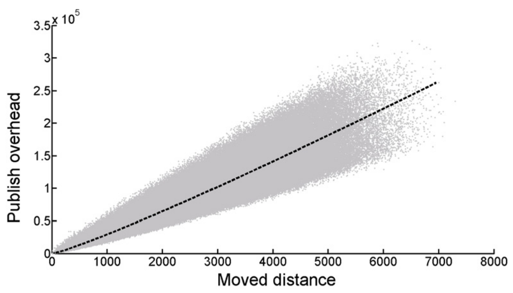

Figure 6: We captured over 600000 node motions during our simulation. This figure shows the publish overhead caused by a moving node plotted against the distance the node moved. The (least-square) average is indicated with the dashed line.

first phase, during which each node moves at least to its first waypoint before any lookup request is issued. Afterwards, the nodes issue lookup requests to randomly chosen nodes, where each node only sends another lookup request when the previous request arrived.

We ran the simulation with 5000 nodes issuing a total of 100000 lookup requests and repeated it for different node speeds and two different world maps. Map_1 corresponds to the world shown in Figure 1. The maximal routing stretch due to lakes is 10, the side-length is 5300 units and ρ was chosen to be 1 unit. Map_2 is of the same size, but contains no lakes at all. The average lookup stretch is shown in the left plot of Figure 5. For increased node speed (1 corresponds to v_{max}^{node}), nodes produce more TFP, which helps lookup requests to find a first pointer to their destination, which in turn decreases the lookup stretch. This fact is supported by the right plot of Figure 5, which shows that lookup requests spend more time following FP and TFP for increased node speed.

In order to obtain the average lookup stretch of approximately 6, we slightly modified the lookup algorithm: Instead of using the slow sequential search for a pointer in the 9 L_i of each level, the sender node s sends the lookup request to all 9 L_i in parallel. If a lookup request finds a pointer to the destination node t, it is immediately forwarded towards t and sends back a `FOUND` message to s. Lookup requests that do not find a pointer to t send back a `NOT_FOUND` message and die. s collects all the responses and only sends the lookup request to the next higher levels L_{i+1} if all responses are negative.

This fast-lookup approach reduces the lookup time by a factor 2 at the cost of increased message overhead. However, more than one copy of the lookup request might be routed towards t if no coordination between the 9 parallel lookup requests takes place. Consequently, each node needs some means to detect messages that were received several times.

For the publish requests, we measured the total message cost dependent on the distance a node moved (Figure 6). Using a least-square approach, we determined the average publish overhead to be $4.3 \cdot d \log d$, where d is the moved distance. This overhead is reasonably small and ensures that the network is not saturated with messages due to moving nodes.

10. CONCLUSIONS

In this paper, we described a location service that supports truly mobile nodes, allowing concurrent routing and mobility. We determined the maximum node speed and proved that `MLS` has constant stretch for lookup requests under the speed constraint.

Open questions remain in the field of bi-directional communication, where nodes might be able to reuse position information from the sender to reply. Also, we used somewhat strong model assumptions (dense node distribution, reliable communication, and the knowledge of the position of lakes). We intend to weaken these constraints in further work.

11. REFERENCES

[1] I. Abraham, D. Dolev, and D. Malkhi. LLS: a Locality Aware Location Service for Mobile Ad Hoc Networks. In *DIALM-POMC*, pages 75–84, 2004.

[2] B. Awerbuch and D. Peleg. Online Tracking of Mobile Users. *J. ACM*, 42(5):1021–1058, 1995.

[3] S. Basagni, I. Chlamtac, V. R. Syrotiuk, and B. A. Woodward. A Distance Routing Effect Algorithm for Mobility (DREAM). In *MOBICOM*, pages 76–84, 1998.

[4] L. Blazevic, J.-Y. Le Boudec, and S. Giordano. A Location-Based Routing Method for Mobile Ad Hoc Networks. *IEEE Trans. Mob. Comput.*, 4(2):97–110, 2005.

[5] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. Routing With Guaranteed Delivery in Ad Hoc Wireless Networks. In *DIAL-M*, pages 48–55, 1999.

[6] F. de Rango, M. Gerla, B. Zhou, and S. Marano. Geo-LANMAR Routing: Asymptotic Analysis of a Scalable Routing Scheme with Group Motion Support. In *BROADNETS*, 2005.

[7] J. Eriksson, M. Faloutsos, and S. Krishnamurthy. Scalable Ad Hoc Routing: The Case for Dynamic Addressing. In *INFOCOM*, 2004.

[8] Q. Fang, J. Gao, and L. J. Guibas. Locating and Bypassing Routing Holes in Sensor Networks. In *INFOCOM*, 2004.

[9] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger. Geometric Ad-Hoc Routing: of Theory and Practice. In *PODC*, pages 63–72, 2003.

[10] F. Kuhn, R. Wattenhofer, and A. Zollinger. Asymptotically Optimal Geometric Mobile Ad Hoc Routing. In *DIAL-M*, 2002.

[11] J.-Y. Le Boudec and M. Vojnovic. Perfect Simulations and Stationarity of a Class of Mobility Models. In *INFOCOM*, 2005.

[12] J. Li, J. Jannotti, D. S. J. D. Couto, D. R. Karger, and R. Morris. A Scalable Location Service for Geographic Ad Hoc Routing. In *MOBICOM*, pages 120–130, 2000.

[13] M. Li, W.-C. Lee, and A. Sivasubramaniam. Efficient Peer-to-Peer Information Sharing over Mobile Ad Hoc Networks. In *MobEA*, 2004.

[14] A. Rao, C. Papadimitriou, S. Shenker, and I. Stoica. Geographic Routing Without Location Information. In *MOBICOM*, pages 96–108, 2003.

[15] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. GHT: a Geographic Hash Table for Data-Centric Storage. In *WSNA*, pages 78–87, 2002.

[16] A. Smith, H. Balakrishnan, M. Goraczko, and N. B. Priyantha. Tracking Moving Devices with the Cricket Location System. In *MobiSys*, 2004.

[17] A. C. Viana, M. D. de Amorim, S. Fdida, Y. Viniotis, and J. F. de Rezende. Easily-Managed and Topology-Independent Location Service for Self-Organizing Networks. In *MobiHoc*, 2005.

[18] S.-C. M. Woo and S. Singh. Scalable Routing Protocol for Ad Hoc Networks. *Wireless Networks*, 7(5):513–529, 2001.

[19] Y. Xue, B. Li, and K. Nahrstedt. A Scalable Location Management Scheme in Mobile Ad-Hoc Networks. In *LCN*, pages 102–111, 2001.

[20] Y. Yu, G.-H. Lu, and Z.-L. Zhang. Enhancing Location Service Scalability with HIGH-GRADE. In *MASS*, 2004.