ABSTRACT
In this work, we present Mobile Matrix, a routing protocol for 6LoWPAN that uses hierarchical IPv6 address allocation to perform any-to-any routing and mobility management without changing a node’s IPv6 address. In this way, device mobility is transparent to the application level. The protocol has low memory footprint, adjustable control message overhead and achieves optimal routing path distortion. Moreover, it does not rely on any particular hardware for mobility detection, such as an accelerometer. Instead, it provides a passive mechanism to detect that a device has moved. We present analytic proofs for the computational complexity and efficiency of Mobile Matrix, as well as an evaluation of the protocol through simulations. Finally, we propose a new mobility model, to which we refer as cyclical random waypoint mobility model, that we use to simulate mobility scenarios, where communication is carried out in environments with limited mobility, such as 6LoWPANs deployed in office buildings, university campuses, concert halls or sports stadiums. Results show that μMatrix delivers 3 times more packets than RPL for top-down traffic over high mobility scenario.

KEYWORDS
Mobility; 6LoWPAN; IPv6; CTP; RPL; any-to-any routing;

1 INTRODUCTION
IPv6 over Low-power Wireless Personal Area Networks (6LoWPAN) is an IETF working group that defines standards for low-power devices to communicate with Internet Protocol. It can be applied even to the small devices to become part of the Internet of Things (IoT). It has defined protocols, including encapsulation and header compression mechanisms, which allow IPv6 packets to be sent and received over low-power devices. These protocols, such as CTP [12] and RPL [24], typically build an acyclic network topology to collect data, such as a tree or a directed acyclic graph. However, they do not handle any-to-any communication or mobility [13].

Mobile Matrix: A Multihop Address Allocation and Any-To-Any Routing in Mobile 6LoWPAN
Bruno P. Santos, Olga Goussevskaia, Luiz F. M. Vieira, Marcos A. M. Vieira, Antonio A.F. Loureiro
Computer Science Department, Universidade Federal de Minas Gerais – Brazil
{bruno.ps,olga,lfvieira,mnvieira,loureiro}@dcc.ufmg.br

Mobility is a major factor present in everyday life. It makes life easier and turns applications more flexible. The usage of many devices for IoT can benefit from it, as is the case of today adoption of smartphones and tablets. By extending IoT protocols to handle mobility, IoT becomes even more ubiquitous.

Matrix (Multihop Address allocation and dynamic any-to-any Routing for 6LoWPAN) [20] is a platform-independent routing protocol for dynamic network topologies and fault-tolerant any-to-any data flows in 6LoWPAN. Matrix uses hierarchical IPv6 address allocation and preserves bi-directional routing.

We present Mobile Matrix (μMatrix), a solution for handling mobility in 6LoWPAN built upon the Matrix protocol. It provides the benefits from Matrix, including any-to-any routing, memory efficiency, reliability, communication efficiency, hardware independence while dealing with mobility efficiently. It enables Matrix to be used in scenarios and applications where mobility is present.

μMatrix handles mobility at the network layer, so the IPv6 address of each node is assigned once and kept unchanged despite mobility. In this way, routing and mobility management is transparent to the application level. The proposed communication protocol has low memory footprint, being suitable for low memory devices, such as wireless sensor networks and IoT. Since there is an intrinsic trade-off between the delay to detect that a node has moved and the number of control messages, μMatrix is able to tune the frequency of control messages according to the application or the mobility pattern. Moreover, μMatrix has optimal routing path distortion, i.e., messages addressed to a mobile node, from anywhere in the network, are sent along the shortest path from the source to its current location, using its original IPv6 address.

To the extent of our knowledge, previous mobile routing protocols for 6LoWPAN have not used hierarchical IPv6 address allocation, but a flat address structure, which incurs in more memory consumption to store the bi-directional routes. On the other hand, protocols for mobile ad hoc networks, like AODV [21] and OLSR [4], have high memory footprint and control message overhead, which makes them not suitable for low power devices or 6LoWPAN.

The main contributions of this paper can be summarized as follows. We present μMatrix, a communication protocol that performs hierarchical IPv6 address allocation and manages routing and mobility without ever changing a node’s IPv6 address. The protocol has low memory footprint, adjustable control message overhead and achieves optimal routing path distortion. We provide analytic proofs for the computational complexity and efficiency of μMatrix, as well as an evaluation of the protocol through simulations. An essential building block of μMatrix is the passive mobility detection mechanism that captures changes in topology without requiring additional hardware (e.g. accelerometer, GPS or compass).
4. Standard routing:

1. Collection tree initialization (Ctree): and then it follows top-down until the destination. a Least Common Ancestor (LCA) between the sender and receiver both previous schemes, i.e., a packet follows bottom-up fashion until a tree that reflects the topology changes due to nodes mobility.

3. Mobility management: Ctree

Matrix builds an address hierarchy tree (IPtree) by using MHCL algorithm [19, 20]. Initially, IPtree has the same topology as underlying data collection protocol (e.g. CTP [12] or RPL [16]) creates a collection routing tree. A tree that reflects the topology changes caused by node mobility.

Moreover, we propose a new mobility model, to which we refer as Cyclic Random Waypoint mobility model (CRWP). In CRWP, nodes are assigned to a home location and might make several moves in random directions, connecting to the 6LoWPAN at different attachment points, and eventually return to their home locations. Our motivation for proposing a new mobility model comes from application scenarios, where communication is carried out in environments with limited mobility, such as 6LoWPANs deployed in an office or school buildings, university campuses or concert halls or sports stadiums.

2 DESIGN OVERVIEW

μMatrix enables any-to-any communication for mobile and static nodes in 6LoWPANs. μMatrix manages mobile nodes without changing its IPv6 address. Also, the protocol preserves all features from previous implementation (such as memory efficiency and fault tolerance) [20]. Figure 1 presents the protocol’s architecture. μMatrix lies at network layer with an underlying data collection protocol (such as CTP or RPL). μMatrix has two planes: i) Control plane able to split and distribute the available address space, manage route tables, and handle mobile nodes; ii) Data plane capable on querying route tables and forward data and control packets.

μMatrix operation consists of the following phases:

1. Collection tree initialization (Ctree): An underlying routing protocol (e.g CTP [12] or RPL [16]) creates a collection routing tree.

2. Descendants convergecast, IPv6 tree: once the collection tree is stable, μMatrix builds an address hierarchy tree (IPv6 tree) by using MHCL algorithm [19, 20]. Initially, IPv6 tree has the same topology as Ctree (top-down direction), but in runtime, they may differ.

3. Mobility management: μMatrix manages the Rtree structure, a tree that reflects the topology changes due to nodes mobility.

4. Standard routing: bottom-up routing follows the Ctree built in phase 1, while top-down the IPv6 tree. Any-to-any routing combines both previous schemes, i.e., a packet flows bottom-up fashion until a Least Common Ancestor (LCA) between the sender and receiver and then it flows top-down until the destination.

2.1 Mobility detection

Mobility detection is a key issue to handle mobile nodes on μMatrix. If nodes by itself inform its motion (e.g. by using accelerometer or GPS) to the protocol, then we refer to as active motion, otherwise if the protocol infer the node movement, we refer to passive motion.

Trickle [17] algorithm passively detects topology changes. However, Trickle lacks in agility to detect changes in dynamic network and mobile nodes. We propose Reverse Trickle timer that operates similarly to the standard algorithm, but in reverse order.

Reverse Trickle introduce a control message and three parameters: i) hasMoved beacon; ii) $I_{\text{max}}$ and $I_{\text{min}}$ the maximum and minimum time interval to send a hasMoved beacon; iii) $k$ the number of attempts to query a node before declaring a inconsistency. These parameters must defined by the network operator before.

Figure 2 shows the Reverse Trickle procedure. First, a node starts sending unicast hasMoved beacons to its parent. $I_{\text{max}}$ is the interval between two consecutive hasMoved. If the node did not receive an ack for a hasMoved beacon, then it sets the interval to $I_{\text{min}}$. After $k$ unsuccessful attempts, the node knows that someone moved. Thus, the node can take actions, for example, properly perform a handover to another parent and then the procedure restarts. Note that by setting the Reverse Trickle parameters, the network operator should consider the trade-off between delay to detect mobility and number of beacons. For a smaller delay to mobility detection, $I_{\text{max}}$ must be tuned to small values at cost of more hasMoved beacons.

In our experiments (Section 5) reverse trickle parameters were set according to application data rate (Table 1).

In [18], the authors argue that a common modification to support mobility is change the control message periodicity. The typical approach uses a simple periodic timer or the standardized Trickle timer. While reverse trickle waits for $I_{\text{max}} + T_{\text{step}}$ to detect a topology change, where $I_{\text{min}} \ll I_{\text{max}}$, the periodic standard-ized Trickle approaches wait for at least $2 \times I_{\text{max}}$.

2.2 Control Plane

2.2.1 Routing data structures. μMatrix maintains three routing trees structures: i) Ctree: a collection tree built by the underlying collection protocol; ii) IPv6 tree: an IPv6 hierarchical tree created by MATRIX initialization and kept static afterward, except when new nodes join the network; iii) Rtree: a tree that reflects the topology changes caused by node mobility.

Initially, IPv6 = Ctree and Rtree = ∅ (see Figures 3(a)(b)). Whenever a topology change occurs due to mobility in Ctree, the
In mobile scenarios, a node fills Mtable upon receiving keepRoute beacons from mobile nodes. The node keeps Mtable entries as long as it receives keepRoute. Otherwise, it uses a THL mechanism to remove entries (see Sec 3.1 for memory footprint analysis). In static scenarios any node stores one-hop neighborhood information in IPParent(η), this requires O(k) entries, where k is the number of node’s children. This memory footprint is better than state-of-the-art, e.g., RPL would need at least 1 routing entry for every child in a node sub-tree for top-down routing fashion.

2.2.2 IPv6 multihop host configuration. μMatrix relies on an underlying collection routing protocol to build the Ctree. Once the Ctree is stable1, the address space available to the border router, e.g., the 64 least-significant bits of the IPv6 address (or a compressed 16-bit representation of the latter), is hierarchically partitioned among nodes in the Ctree. The (top-down) address distribution is preceded by a (bottom-up) convergecast phase, in which each node counts the total number of its descendants and propagates it to its parent. Thus node knows how many descendants each child has. Such information is required to distribute IP ranges in a fairly way. As result of this procedure is obtained the IPtree.

Figure 4 most left shows the process. First, it is created the Ctree (upwards arrows), and then, after the Ctree stabilization, the convergecast phase occurs allowing nodes to know the size of theirs sub-tree (%) next to each node). Next, the root starts the IP distribution by auto-setting its IP (e.g. the first available IP from range) and then reserving a portion of the range for delayed nodes. Next, the node distributes the remaining range fairly between its children (e.g. in Figure 4 B receives 70% of available range, i.e., from 16 to 183). Finally, each node repeats the IP distribution process.

2.2.3 Mobility management. After host configuration, μMatrix starts the mobile engine allowing nodes to move around the 6LoWPAN. Mobile engine uses a finite state machine (Figure 5). Each node can be in one state depending on its previous condition and the knowledge about its neighborhood. The engine also uses Reverse Trickle to recognize mobility and transit among states. In the following, we discuss the actions taken in each of those states.

\[1A \text{node is stable if it reaches } k \text{ times the maximum maintenance beacon period of Ctree protocol without changing its parent. We use Trickle [17] as beacon scheme.}\]
Each node begins at Home Location (HL) state. In HL, the nodes start the reverse trickle with its CT\textit{parent}(\eta) (initially, CT\textit{parent}(\eta) = IP\textit{parent}(\eta) see Figure 6(a)). When reverse trickle identifies a mobility event, then the node transit to Someone Moved (SM) state.

When a node is at SM state, it knows that someone moved, but it does not know if itself or its parent moved. There are at least two ways to automatically find out who moved. First, a node proactively queries its children (IP\textit{Children}(\eta)), if no one answer then the node moved; otherwise the parent moved. Second, a node must wait for period (e.g. one \textit{I\textsubscript{max}}) to receives \textit{hasMoved} beacons from its children and then infer who moved. We use the second approach in our implementation. After discovering who moved, the node goes to Node Moved (NM) or Parent Moved (PM) state.

Several actions are taken when a node reaches NM state. Firstly, the node disables the IP\textit{Children}(\eta) table due to the node new position in the Ctree. Next, the \textit{Mtable} is cleaned, because it must be outdated. Then, the node triggers the new parent discovery from underlying collection protocol. When the node is attached again to the Ctree, it restarts reverse trickle with new CT\textit{parent} and begins sending keepRoute. IP\_ONLY at a frequency of $\delta$ to its IP\textit{parent}. Figure 6(a)(b) shows this situation. When B sends keepRoute beacons to A, when B moves and finds a new CT\textit{parent}. The beacons travel upward to the LCA(A, B) and then downwards to the node A.

When a node reaches PM, this means that its parent moved. Then, the node triggers the parent discovery mechanism. When it is attached again to Ctree, it restarts the reverse trickle and begins sending two keepRoute beacons (if it has children the beacon contains IP\_AND\_RANGE, otherwise IP\_ONLY at a frequency of $\delta$ to IP\textit{parent} and its grand IP\textit{parent}. Figure 6(c)(d) illustrate this situation. If B moves, then C eventually reaches PM state, and then C begins sending keepRoute beacons to its grandIP\textit{parent} = A. The messages travel to LCA(A, C) and then to the ultimate destinations.

Eventually, nodes return to their home position being attached again to its IP\textit{parent} in Ctree. This situation also triggers some actions. First, the node stops to sending keepRoute beacons and sends to its very previous CT\textit{parent} = PRV\textit{parent} a rmRoute containing its information to properly remove outdated routes nodes’ \textit{Mtable}. Also, the returned node restarts the reverse trickle with its IP\textit{parent}.

Optional features are made to improve the mobile node management. Note that if a node is attached to a sequence of CT\textit{parent} before returning to home location, then several states will be installed in the network. Although the \textit{Mtable} THL field exists to remove inconsistent entries, it is possible to send rmRoute beacons to each node’ PRV\textit{parent} to eliminate such inconsistency.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig5.png}
\caption{Mobile Engine state machine.}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig6.png}
\caption{Mobile engine operation after mobility events.}
\end{figure}

Discussion: note that a sub-tree can move and nodes still hear each other. For instance, in Figure 6(c) suppose A and B move together. Then, A and C eventually will transit to PM, while B and C’s sub-tree remain in HL. In this case, the LCA has two \textit{Mtable} entries matching with C’s sub-tree, but one more restrictive than other. Thus, the LCAs play a key role, which they always route through the most restrictive \textit{Mtable} range match available.

Also, note that \textit{µMatrix} preserves locality when manages mobile nodes since no \textit{MTables} needs updates above LCA. Figure 4 (right-most) illustrates this situation. When node B moves, it transits to NM state, \textit{Mtables} comprised in the path B to A = IP\textit{parent}(B) = LCA(B, IP\textit{parent}(B)) receive updates. B movement causes E and F to transit to PM state. Then, E and F find new routes and update the \textit{Mtables} between them and its grandIP\textit{parent} = A. Note that \textit{Mtable}(C) requires only one entry for both E and F sub-trees because their IP range are contiguous and an aggregation was made.

2.2.4 Loop avoidance. Dynamic links and mobile nodes cause topology route information to become outdated, which causes routing loops [12]. \textit{µMatrix} uses data path validation and adaptive beaconing to detect loops CTP and RPL [12, 16]. Besides that, if a node receives more than one unique control packet\footnote{Together the keepRoute fields (see Sec 2.2.1) denote a unique packet instance.} in a short time, then this indicates an inconsistency in the tree, which triggers the
control packet suppression and the underlying protocol route update. Also, Mtable and keepRoute beacon have respectively Time Has Lived (THL) and Time To Live (TTL) field, which is used to remove inconsistent routes and messages from the network.

2.3 Data Plane: any-to-any routing
The Forwarding Engine (see Figure 1) is responsible for data forwarding. Any-to-any routing combines bottom-up forwarding, until the LCA between the sender and receiver, and then top-down forwarding to the destination. Upon receiving a data packet, the node checks if the message is for itself. Second, the node tries to match the destination with an entry Mtable. Third, if any Mtable entries match with the target address, then the node checks if the packet destination falls within some range in \( r \in IP\text{children}(\eta) \), if positive match, then the node forwards the packet downwards. If not, then the node forwards the packet downwards according. Finally, if all previous attempts fail, then the node sends the packet upwards using CTparent(\eta).

3 COMPLEXITY ANALYSIS
For the formal analysis, we assume a synchronous communication message-passing model with no faults. Thus, all nodes start executing the algorithm simultaneously and the time is divided into synchronous rounds, i.e., when a message is sent from node \( v \) to its neighbor \( u \) in time-slot \( t \), it must arrive at \( u \) before time-slot \( t + 1 \), and \( d(v, u) \) is the shortest path length between \( v \) and \( u \) in Ctree \( \cup IP\text{tree} \cup RC\text{tree} \). The performance of \( \mu \text{Matrix} \) in faulty scenarios is analyzed through simulations in Section 5.

3.1 Memory footprint
As described in Section 2, the temporary routing information needed to maintain mobility is stored in the Mtable data structure of some nodes. Each entry is kept for at most \( TTL_{\text{max}} \) seconds, a time interval pre-configured by the network operator, and is deleted unless a keepRoute beacon is received. In the following theorem, we bound the total number of Mtable entries in the network, necessary to manage routing of each mobile node \( \mu \in C\text{tree} \).

**Theorem 3.1.** The memory footprint to manage the mobility of one node \( \mu \in C\text{tree} \) with \( \mu \text{Matrix} \) is \( M(\mu) = O(\text{depth}(C\text{tree})) \).

Proof. Consider a node \( \mu \in C\text{tree} \) that has moved from its home location in time-slot \( t_0 \) and returned in time-slot \( t_f \). Consider the (permanent) IPparent(\( \mu \)) and the (temporary) CTparent(\( \mu \)) in time-slot \( t_0 < t_1 < t_f \). A routing entry for the temporary location of \( \mu \) will be stored in the Mtable of every node comprising the shortest path between IPparent(\( \mu \)) and CTparent(\( \mu \)). Moreover, if \( \mu \) has descendants in IP\text{tree}, i.e, \( k(\mu) = | IP\text{children}(\mu) | > 0 \), then each node \( c \in IP\text{children}(\mu) \) will send a temporary (bi-directional) route request to their respective CTparent(\( c \)), and a (temporary) routing entry will be stored in the Mtable of every node comprising the shortest path between CTparent(\( c \)) and IPparent(\( \mu \)). Therefore, the total memory footprint to manage the mobility of a node \( \mu \) is:

\[
M(\mu) = d(CTparent(\mu), IPparent(\mu)) + 1 + \sum_{c \in IPchildren(\mu)} (d(CTparent(c), IPparent(\mu)) + 1) \\
\leq (k(\mu) + 1) \times (\text{depth}(C\text{tree}) + 1) \\
= O(\text{depth}(C\text{tree}))
\]

Theorem 3.1 implies that the total memory footprint to manage the mobility of \( m \) nodes is \( O(m \times \text{depth}(C\text{tree})) \). Note that \( \mu \text{Matrix} \) preserves locality when managing mobile routing information of a node \( \mu \), since no Mtable needs to be updated at nodes above the LCA(IPparent(\( \mu \)), CTparent(\( \mu \))).

3.2 Control message overhead
Control messages used by \( \mu \text{Matrix} \) are comprised of three types: (1) those used by Matrix to set up the initial IPtree and address allocation; (2) hasMoved beacons, defined in Section 2.1; and (3) keepRoute beacons, defined in Section 2.2.1.

For any network of size \( n \) with a spanning collection tree Ctree rooted at node \( r \), the message and time complexity of Matrix protocol in the address allocation phase is \( \text{Msg}(\mu \text{Matrix}(IP\text{tree})) = O(n) \) and \( T(\mu \text{Matrix}(IP\text{tree})) = O(\text{depth}(C\text{tree})) \), respectively, which is asymptotically optimal, as proved in [20]. Next we bound the number of control messages of type (2) and (3).

**Theorem 3.2.** Consider a network with \( n \) nodes, with a spanning collection tree Ctree rooted at node \( r \), and \( m \) mobility events, consisting of \( m \) nodes \( \mu_i \), changing location during time intervals \( \Delta_i \leq \Delta \) time-slots. Moreover, consider the hasMoved beacon parameters \( l_{\text{min}} \), \( l_{\text{max}} \) and \( l_k \) and the keepRoute beacon interval of \( \delta \) time-slots. The control message complexity of \( \mu \text{Matrix} \) to perform routing under mobility of \( m \) nodes is

\[
\text{Msg}(\mu \text{Matrix}(C\text{tree})) = O \left( \frac{m \times l_k}{l_{\text{min}}} + \frac{n}{l_{\text{max}}} \right) + O \left( \frac{m \times \Delta}{\delta \times \text{depth}(C\text{tree})} \right).
\]

Proof. Firstly, we bound the number of hasMoved beacons, which are sent periodically by all nodes in order to detect mobility events. As described in Section 2.1, when there is no mobility, the periodicity of hasMoved beacons is \( 1/l_{\text{max}} \). If some node \( \mu \) has moved (an ack is lost), then \( l_k \) messages are sent in intervals of \( l_{\text{min}} \), \( 1/l_{\text{min}} \) time-slots. Using the fact that the network is a tree and the number of edges is \( O(n) \), this gives a total of messages

\[
\text{Msg}(\mu \text{Matrix}(C\text{tree})) = O \left( \frac{m \times l_k}{l_{\text{min}}} + \frac{n}{l_{\text{max}}} \right).
\]

Now, we bound the number of keepRoute beacons. As described in Section 2, mobile nodes send periodic keepRoute beacons at a frequency of \( \delta \) to keep the Mtables up-to-date. Consider a node \( \mu \) in Ctree that has moved from its home location in time-slot \( t_0 \) and returned in time-slot \( t_f \). Consider the IPparent(\( \mu \)), CTparent(\( \mu \)) in time-slot \( t_0 < t_1 < t_f \), and \( \Delta = t_f - t_0 \). When \( \mu \) is attached to a CTparent(\( \mu \)), \( \mu \) sends keepRoute beacons at a rate of \( \delta \) for at most \( \Delta \) time-slots, such beacons travel the shortest path \( || CTparent(\mu), IPparent(\mu) || \leq 2 \times \text{depth}(C\text{tree}) \). Furthermore, if
μ has descendants, i.e., k(μ) = |IPchildren(μ)| > 0, then each node c ∈ IPchildren(μ) will also send keepRoute beacons at a rate of δ for at most δ time-slots, such beacons will travel the shortest path |CTparent(c), IPparent(μ)| ≤ 2 × depth(CTree). Therefore, the total control overhead to manage the mobility of a node μ is at most 2 × depth(CTree)(k(μ) + 1)/δ, which results in

\[ \text{Msg}(μ) = \frac{m \times \Delta}{\delta} \text{depth}(\text{CTree}). \]

Finally, the total control overhead is bounded by:

\[ \text{Msg}(μ) = \text{Msg}(μ) + \text{Msg}(μ). \]

Once again, μMatrix preserves locality when managing mobile routing state of a node μ since no messages need to be sent to nodes above the LCA(IPparent(μ), CTparent(μ)).

### 3.3 Routing path distortion

We analyze the route length of messages, addressed to mobile nodes. Consider the underlying collection protocol (e.g., CTP or RPL), which dynamically optimizes the (bottom-up, or upwards) links in the collection tree CTree, according to some metric, such as ETX. We define an optimal route length as the distance of the shortest path between (s, d), comprised of the upwards links of the collection tree CTree and the downwards links of the union of the IPv6 address tree and the reverse-collection tree, i.e., IPtree ∪ RCtree.

**Theorem 3.3.** μMatrix presents optimal path distortion under mobility, i.e., all messages are routed along shortest paths towards mobile destination nodes.

**Proof.** Consider a mobile node μ ∈ CTree, which has moved from its home location in time-slot t0. Messages addressed to μ and originated by some node η ∈ CTree in time-slot t1 > t0 can belong to traffic flows of three kinds: (1) bottom-up: LCA1(μ, η) = μ; (2) top-down: LCA1(μ, η) = η; and (3) any-to-any: LCA1(μ, η) ≠ μ ≠ η. In case (1), messages are forwarded using the underlying collection protocol, using the upwards links of the collection tree CTree, which is optimal. In case (2), messages are forwarded using MTables of η and its descendents, until reaching the mobile location of μ in some time-slot tf > t0. This path is comprised of the downwards links of IPtree ∪ RCtree in time-slot t0 < t1 ≤ tf, which is the optimal route from η to the mobile location of μ in that time-slot. In case (3), the route between η and LCA1(μ, η) falls into the case (1) and the route between LCA1(μ, η) and μ falls into the case (2), for some t0 < t1 ≤ tf. Theorem 3.3 is optimal.

### 4 CRWP MOBILITY MODEL

Here, we propose the Cyclical Random Waypoint Mobility Model (CRWP), a mobility model based on the Random Waypoint [2]. CRWP is useful to model scenarios where the entities move to different destinations, and eventually, they return to their initial positions. Which is the case of people and their portable devices in offices, universities, hospitals, factories, etc.

In CRWP, the entities move independently to random destinations and speeds as in RWP. When an entity arrives at the destination, it stops for a given time Tpause. A difference in CRWP is that after n chosen destinations, the mobile entity returns to its initial position. Besides that, only k% of mobile entities are outside of their initial position in each instant of time. CRWP has four parameters: i) PerMobNodes: maximum percentage of entities that are mobile in each instant of time; ii) Stopped: number of stops that the mobile entity do before returning to its original position; iii) Speed: speed which the mobile entity moves; iv) Tpause: the amount of time that the entity stays in a destination position.

### 5 SIMULATION RESULTS

We implement μMatrix as a subroutine of collection protocol available in ContikiOS [7] and the experiments were run on Cooja [9]. We compare μMatrix with ContikiOS RPL implementation. We use the BonnMotion [1] to implement CRWP as well as to generate and analyze mobility traces. We simulated four different scenarios. The first scenario represents the static network, in which nodes do not move. The remaining represent mobility scenarios named low, moderate, and high with mobile nodes. Table 1 lists the default simulation parameters used for each scenario.

On top of the network layer, we ran an application, in which each node sends 20 data packets to the root. Upon receiving a data packet, the root confirms to the sender with an ack packet that has the size of a data packet. The application waits for 10 min for protocols initialization and stabilization before it starts sending data. The nodes start sending their data in a simulation time randomly chosen in (10, 20] min. The mobility traces were configured to start after the stabilization time. Additionally, we generate 10 mobility traces for each scenario. Each trace and the static scenario were run 10 times, totaling 3010 runs. In each plot, the bars represent the average, and the error bars the confidence interval of 95%, and the curves are the maximum table usage for a given mobility scenario. **Mobility scenario:** We simulated a scenario, where n = 100 people are assumed to be in an office and can move around and return to a predefined home position. This scenario is expected to present relatively low mobility, thus in our set up, k% of the nodes are moving at any moment in time, where k ∈ {5, 10, 15}. Table 2 presents some mobility metrics [1] for each scenario. We highlight that link breaks play a key role in the performance of the network.
we set variability, they reflect the simulation parameters, where the node lack of memory (see Figure 7) to store top-down routes.

More mobility is allowed. In the harshest mobility scenario, the \( \mu \) rate. In mobility scenarios, that, when there is no mobility, bottom-up routes are rapidly rebuilt, and the reliability increases. Triggers the underlying route discovery, and as a consequence, protocols (the total number of beacons sent during the entire simulation). RPL sends fewer control packets than \( \mu \)Matrix beacons. \( \mu \)Matrix allows tuning the Reverse Trickle fire rate to reduce the sending beacons, but note that the adjustment reverse trickle faces a trade-off between quick mobility discovery and control overhead. In Table 1, we set \( I_{\text{max}} \) of RPL and \( \mu \)Matrix evenly and close to data packet rate, which gives to the protocols the fair opportunity to identify topology changes and react to them.

Table 2: Mobility Metrics

<table>
<thead>
<tr>
<th>Mobility Metrics</th>
<th>Low Mob. sce.</th>
<th>Mod. Mob. sce.</th>
<th>High Mob. sce.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. Link Breaks</td>
<td>1623</td>
<td>3057</td>
<td>4838</td>
</tr>
<tr>
<td>Avg. Link duration</td>
<td>761.90</td>
<td>457.4</td>
<td>345</td>
</tr>
<tr>
<td>Avg. Degree</td>
<td>4.12</td>
<td>4.36</td>
<td>4.44</td>
</tr>
<tr>
<td>Avg. Time to link break</td>
<td>227.6</td>
<td>216.1</td>
<td>204.5</td>
</tr>
</tbody>
</table>

5.1 Results

In Figure 7, we show the Cumulative Distribution Functions (CDFs) of the percentage of downward routing table usage among nodes for given mobility scenario. In static scenarios, all \( \mu \)Matrix nodes use up to 25% of available downwards route entries, while RPL < 75% of nodes use up to 25% of entries. Indeed, for some RPL nodes, 100% of table entries are used. Usually, those nodes that use more memory are near to the root, and they play a fundamental role in top-down routing. If they have a full downward routing table, then the traffic pattern top-down suffers from poor reliability, and some nodes may be unreachable. In mobility scenarios, \( \mu \)Matrix also presents more efficient memory footprint, and the difference grows up in high mobility scenarios, where > 50% of RPL nodes have all table entries busy, while \( \mu \)Matrix nodes use at most 70% of downwards available routes.

Figure 8(a) shows the amount of control traffic overhead of the protocols (the total number of beacons sent during the entire simulation). RPL sends fewer control packets than \( \mu \)Matrix, but the difference between them does not exceed 7.4%. \( \mu \)Matrix sends more beacons to react to topology changes quickly. Reverse Trickle is responsible for firing most of \( \mu \)Matrix beacons. \( \mu \)Matrix allows tuning the Reverse Trickle fire rate to reduce the sending beacons, but note that the adjustment reverse trickle faces a trade-off between quick mobility discovery and control overhead. In Table 1, we set \( I_{\text{max}} \) of RPL and \( \mu \)Matrix evenly and close to data packet rate, which gives to the protocols the fair opportunity to identify topology changes and react to them.

Packets Reception Rate (PRR) is a metric of network reliability. It computes the number of packets received successfully over all packets sent. Figure 8(b) shows the PRR in bottom-up data traffic. In all scenarios, \( \mu \)Matrix presents higher PRR rate than RPL. When \( \mu \)Matrix realizes that a topological change happened, it quickly triggers the underlying route discovery, and as a consequence, bottom-up routes are rapidly rebuilt, and the reliability increases.

Figure 8(c) shows the PRR for top-down data traffic. We can see that, when there is no mobility, \( \mu \)Matrix presents 99.9% of success rate. In mobility scenarios, \( \mu \)Matrix PRR decreases slowly when more mobility is allowed. In the harshest mobility scenario, the PRR > 75%. RPL, on the other hand, suffer from poor reliability, delivering < 21.1% in all simulated scenarios, which occurs due to the lack of memory (see Figure 7) to store top-down routes.

6 RELATED WORK

In the world of tiny (IoT) several mobility-enabling routing protocols have been proposed. Firstly we highlight \( \mu \)Matrix’s features against its original static version [20]. Then, we survey recent protocols in the context of 6LoWPAN and put them in perspective with \( \mu \)Matrix.

Matrix was originally proposed without support for mobility[20]. If a node moved from its home location, the hierarchical IPv6 address allocation would become invalid and compromise downward routing. Although RPL [24] is the standard protocol for 6LoWPANs, it presents limitations, for example, in mobility scenarios, scalability issues, reliability and robustness for point-to-multipoint traffic [13, 20]. Most recent mobile-enabled routing protocols are RPL extensions. They deal with mobile issues, but they do not handle RPL drawbacks. Co-RPL [11] provides mobility support to RPL but without Trickle. This turns Co-RPL more responsive but has higher overhead. MMRPL [5] modifies the RBL beacon periodicity by replacing the Trickle mechanism with a Reverse Trickle-Like. Their Reverse Trickle decays exponentially, while our approach quickly goes to the minimum after an unacknowledged beacon. MMRPL also needs some static nodes. In ME-RPL [8], static nodes have higher priority than mobile ones. ME-RPL requires some fixed nodes. The memory requirement to downward routes is still prohibitive. mRPL [10] proposes a hand-off mechanism for mobile nodes in RPL by separating nodes into mobile (MN) or serving access point (AP). They use smart-HOP algorithm on MN nodes to perform hand-off between AP.

XCTP [23] extends CTP to support bidirectional traffic. XCTP does not support IPv6 addressing and any-to-any traffic. Hydro [6] fills the gap of any-to-any traffic, but it requires static nodes with a large memory to perform the routing and support mobility nodes.
Mobile IP [22] and Hierarchical Mobile IPv6 (HIPv6) Mobility Management [3] are standards for IPv6 networks for handling local mobility. However, they are not designed for 6LoWPANs, as they do not present a mobility detection or adjustable timers. LOAD [15] and DYMO-Low [14] are 6LoWPAN protocols inspired in AODV and DYMO, but they are not suitable for mobile networks.

Table 3 summarizes properties of the related protocols.

<table>
<thead>
<tr>
<th>Feature</th>
<th>μMatrix</th>
<th>RPL</th>
<th>CoRPL</th>
<th>MRPL</th>
<th>SIG-6</th>
<th>mRPL</th>
<th>DMR</th>
<th>Hybla</th>
<th>XTP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retransmit</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Top-down</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Any-to-any</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Address Allocation</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Resend</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Memory efficiency</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Fault Tolerance</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>IPv6 Support</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Address Allocation</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Topological Changes</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Constraints</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

Figure 8: Simulation experiments

Table 3: Routing protocol properties

7 CONCLUSIONS

In this work, we presented μMatrix: a memory efficient routing protocol for 6LoWPAN that performs any-to-any routing, hierarchical address allocation, and mobility management. As a building block of μMatrix, we proposed a passive mobility detection mechanism that captures topological changes without requiring additional hardware. Finally, we introduced the CRWP, a mobility model suited for scenarios with mobile nodes that have cyclical movement patterns. As future work, we plan to run experiments with physical devices and extend experimental evaluation to more mobile models, such as faulty communications scenarios.

ACKNOWLEDGEMENTS

We thank the research agencies CAPES, CNPq and FAPEMIG.

REFERENCES